Colour Documentation

Release 0.3.15

Colour Developers

Jan 25, 2020

CONTENTS

1 2 Sponsors **3**

2 3 Features **5**

3 4 Installation **7**

4 5 Documentation **9**

4.1 5.1 Tutorial . . . 9
4.2 5.2 How-To Guide . . . 9
4.3 5.3 API Reference . . . 9
4.3.1 Colour Manual . . . 9
4.3.1.1 Tutorial . . . 9
4.3.1.2 Basics . . . 31
4.3.1.3 Advanced . . . 35
4.3.1.4 Reference . . . 36
4.3.1.5 Bibliography . . . 715
4.4 5.4 Jupyter Notebooks . . . 716
4.5 5.5 Examples . . . 716
4.5.1 5.5.1 Automatic Colour Conversion Graph - `colour.graph` . . . 716
4.5.2 5.5.2 Chromatic Adaptation - `colour.adaptation` . . . 717
4.5.3 5.5.3 Algebra - `colour.algebra` . . . 717
4.5.3.1 5.5.3.1 Kernel Interpolation . . . 717
4.5.3.2 5.5.3.2 Sprague (1880) Interpolation . . . 717
4.5.4 5.5.4 Colour Appearance Models - `colour.appearance` . . . 717
4.5.5 5.5.5 Colour Blindness - `colour.blindness` . . . 718
4.5.6 5.5.6 Colour Correction - `colour characterisation` . . . 718
4.5.7 5.5.7 Colorimetry - `colour.colorimetry` . . . 718
4.5.7.1 5.5.7.1 Spectral Computations . . . 718
4.5.7.2 5.5.7.2 Multi-Spectral Computations . . . 718
4.5.7.3 5.5.7.3 Blackbody Spectral Radiance Computation . . . 719
4.5.7.4 5.5.7.4 Dominant, Complementary Wavelength & Colour Purity Computation . . . 719
4.5.7.5 5.5.7.5 Lightness Computation . . . 719
4.5.7.6 5.5.7.6 Luminance Computation . . . 720
4.5.7.7 5.5.7.7 Whiteness Computation . . . 720
4.5.7.8 5.5.7.8 Yellowness Computation . . . 720
4.5.7.9 5.5.7.9 Luminous Flux, Efficiency & Efficacy Computation . . . 720
4.5.8 5.5.8 Contrast Sensitivity Function - `colour.contrast` . . . 721
4.5.9 5.5.9 Colour Difference - `colour.difference` . . . 721
4.5.10 5.5.10 IO - `colour.io` . . . 721
4.5.10.1 5.5.10.1 Images . . . 721
4.5.10.2 5.5.10.2 Look Up Table (LUT) Data . . . 721
4.5.11 5.5.11 Colour Models - `colour.models` . . . 722
4.5.11.1 5.5.11.1 CIE xyY Colourspace . . . 722

4.5.11.2 5.5.11.2 CIE L*a*b* Colourspace . . . 722
4.5.11.3 5.5.11.3 CIE L*u*v* Colourspace . . . 722
4.5.11.4 5.5.11.4 CIE 1960 UCS Colourspace . . . 722
4.5.11.5 5.5.11.5 CIE 1964 U*V*W* Colourspace . . . 722
4.5.11.6 5.5.11.6 Hunter L,a,b Colour Scale . . . 722
4.5.11.7 5.5.11.7 Hunter Rd,a,b Colour Scale . . . 722
4.5.11.8 5.5.11.8 CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006) . . . 723
4.5.11.9 5.5.11.9 CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017) . . . 723
4.5.11.10 5.5.11.10 IPT Colourspace . . . 723
4.5.11.11 5.5.11.11 DIN99 Colourspace . . . 723
4.5.11.12 5.5.11.12 hdr-CIELAB Colourspace . . . 723
4.5.11.13 5.5.11.13 hdr-IPT Colourspace . . . 723
4.5.11.14 5.5.11.14 OSA UCS Colourspace . . . 724
4.5.11.15 5.5.11.15 JzAzBz Colourspace . . . 724
4.5.11.16 5.5.11.16 Y'CbCr Colour Encoding . . . 724
4.5.11.17 5.5.11.17 YCoCg Colour Encoding . . . 724
4.5.11.18 5.5.11.18 ICTCP Colour Encoding . . . 724
4.5.11.19 5.5.11.19 HSV Colourspace . . . 724
4.5.11.20 5.5.11.20 Prismatic Colourspace . . . 724
4.5.11.21 5.5.11.21 RGB Colourspace and Transformations . . . 724
4.5.11.22 5.5.11.22 RGB Colourspace Derivation . . . 725
4.5.11.23 5.5.11.23 RGB Colourspaces . . . 725
4.5.11.24 5.5.11.24 OETFs . . . 726
4.5.11.25 5.5.11.25 OETFs Inverse . . . 726
4.5.11.26 5.5.11.26 EOTFs . . . 726
4.5.11.27 5.5.11.27 EOTFs Inverse . . . 727
4.5.11.28 5.5.11.28 OOTFs . . . 727
4.5.11.29 5.5.11.29 OOTFs Inverse . . . 727
4.5.11.30 5.5.11.30 Log Encoding / Decoding . . . 727
4.5.11.31 5.5.11.31 CCTFs Encoding / Decoding . . . 728
4.5.12 5.5.12 Colour Notation Systems - `colour.notation` . . . 728
4.5.12.1 5.5.12.1 Munsell Value . . . 728
4.5.12.2 5.5.12.2 Munsell Colour . . . 729
4.5.13 5.5.13 Optical Phenomena - `colour.phenomena` . . . 729
4.5.14 5.5.14 Light Quality - `colour.quality` . . . 729
4.5.14.1 5.5.14.1 Colour Rendering Index . . . 729
4.5.14.2 5.5.14.2 Colour Quality Scale . . . 729
4.5.14.3 5.5.14.3 Academy Spectral Similarity Index (SSI) . . . 729
4.5.15 5.5.15 Spectral Up-sampling & Reflectance Recovery - `colour.recovery` . . . 730
4.5.16 5.5.16 Correlated Colour Temperature Computation Methods - `colour.temperature` . . . 730
4.5.17 5.5.17 Colour Volume - `colour.volume` . . . 730
4.5.18 5.5.18 Plotting - `colour.plotting` . . . 730
4.5.18.1 5.5.18.1 Visible Spectrum . . . 730
4.5.18.2 5.5.18.2 Spectral Distribution . . . 731
4.5.18.3 5.5.18.3 Blackbody . . . 732
4.5.18.4 5.5.18.4 Colour Matching Functions . . . 732
4.5.18.5 5.5.18.5 Luminous Efficiency . . . 733
4.5.18.6 5.5.18.6 Colour Checker . . . 734
4.5.18.7 5.5.18.7 Chromaticities Prediction . . . 735
4.5.18.8 5.5.18.8 Colour Temperature . . . 736
4.5.18.9 5.5.18.9 Chromaticities . . . 737
4.5.18.10 5.5.18.10 Colour Rendering Index . . . 738

5 6 Contributing **741**

6 7 Changes 743
7 8 Bibliography 745
8 9 See Also 747
9 10 Code of Conduct 749
10 11 About 751
Bibliography 753
Index 769

Colour is an open-source Python package providing a comprehensive number of algorithms and datasets for colour science.

It is freely available under the New BSD License terms.

Colour is an affiliated project of NumFOCUS, a 501(c)(3) nonprofit in the United States.

Table of Contents

- *1 Draft Release Notes*
- *2 Sponsors*
- *3 Features*
- *4 Installation*
- *5 Documentation*
  - *5.1 Tutorial*
  - *5.2 How-To Guide*
  - *5.3 API Reference*
  - *5.4 Jupyter Notebooks*
  - *5.5 Examples*
    - *5.5.1 Automatic Colour Conversion Graph* - `colour.graph`
    - *5.5.2 Chromatic Adaptation* - `colour.adaptation`
    - *5.5.3 Algebra* - `colour.algebra`
    - *5.5.4 Colour Appearance Models* - `colour.appearance`
    - *5.5.5 Colour Blindness* - `colour.blindness`
    - *5.5.6 Colour Correction* - `colour characterisation`
    - *5.5.7 Colorimetry* - `colour.colorimetry`
    - *5.5.8 Contrast Sensitivity Function* - `colour.contrast`
    - *5.5.9 Colour Difference* - `colour.difference`
    - *5.5.10 IO* - `colour.io`

        * *5.5.11 Colour Models* - `colour.models`
        * *5.5.12 Colour Notation Systems* - `colour.notation`
        * *5.5.13 Optical Phenomena* - `colour.phenomena`
        * *5.5.14 Light Quality* - `colour.quality`
        * *5.5.15 Spectral Up-sampling & Reflectance Recovery* - `colour.recovery`
        * *5.5.16 Correlated Colour Temperature Computation Methods* - `colour.temperature`
        * *5.5.17 Colour Volume* - `colour.volume`
        * *5.5.18 Plotting* - `colour.plotting`
- *6 Contributing*
- *7 Changes*
- *8 Bibliography*
- *9 See Also*
- *10 Code of Conduct*
- *11 About*

The draft release notes from the develop branch are available at this url.

CHAPTER

ONE

2 SPONSORS

We are grateful for the support of our sponsors. If you'd like to join them, please consider becoming a sponsor on OpenCollective.

CHAPTER

TWO

3 FEATURES

Colour features a rich dataset and collection of objects, please see the features page for more information.

CHAPTER

THREE

4 INSTALLATION

Colour can be easily installed from the Python Package Index by issuing this command in a shell:

```
$ pip install colour-science
```

Colour is also available for Anaconda from *Continuum Analytics* via conda-forge:

```
$ conda install -c conda-forge colour-science
```

The detailed installation procedure is described in the Installation Guide.

CHAPTER

FOUR

5 DOCUMENTATION

4.1 5.1 Tutorial

The static tutorial provides an introduction to **Colour**. An interactive version is available via Google Colab.

4.2 5.2 How-To Guide

The How-To guide for **Colour** shows various techniques to solve specific problems and highlights some interesting use cases.

4.3 5.3 API Reference

The main technical reference for **Colour** and its API is the Colour Manual.

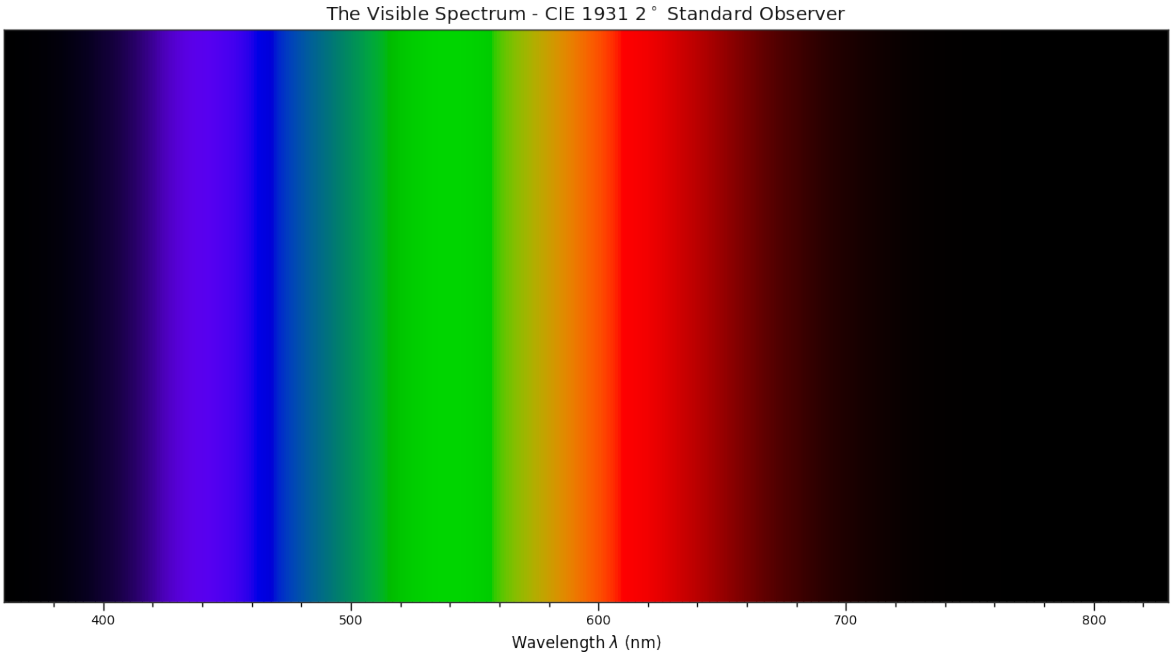
4.3.1 Colour Manual

4.3.1.1 Tutorial

Colour spreads over various domains of Colour Science, from colour models to optical phenomena, this tutorial does not give a complete overview of the API but is a good introduction to the main concepts.

Note: A directory with examples is available at this path in **Colour** installation: *colour/examples*. It can also be explored directly on Github.

```
from colour.plotting import *

colour_style()

plot_visible_spectrum()
```



Overview

Colour is organised around various sub-packages:

- *adaptation*: Chromatic adaptation models and transformations.
- *algebra*: Algebra utilities.
- *appearance*: Colour appearance models.
- *biochemistry*: Biochemistry computations.
- *blindness*: Colour vision deficiency models.
- *continuous*: Base objects for continuous data representation.
- *contrast*: Objects for contrast sensitivity computation.
- *characterisation*: Colour fitting and camera characterisation.
- *colorimetry*: Core objects for colour computations.
- *constants*: *CIE* and *CODATA* constants.
- *corresponding*: Corresponding colour chromaticities computations.
- *difference*: Colour difference computations.
- *examples*: Examples for the sub-packages.
- *graph*: Graph for automatic colour conversions.
- *io*: Input / output objects for reading and writing data.
- *models*: Colour models.
- *notation*: Colour notation systems.
- *phenomena*: Computation of various optical phenomena.
- *plotting*: Diagrams, figures, etc. . .
- *quality*: Colour quality computation.
- *recovery*: Reflectance recovery.

- *temperature*: Colour temperature and correlated colour temperature computation.
- *utilities*: Various utilities and data structures.
- *volume*: Colourspace volumes computation and optimal colour stimuli.

Most of the public API is available from the root `colour` namespace:

```
import colour

print(colour.__all__[:5] + ['...'])
```

```
['domain_range_scale', 'get_domain_range_scale', 'set_domain_range_scale', 'CHROMATIC_ADAPTATION_
↪METHODS', 'CHROMATIC_ADAPTATION_TRANSFORMS', '...']
```

The various sub-packages also expose their public API:

```
from pprint import pprint

for sub_package in ('adaptation', 'algebra', 'appearance', 'biochemistry',
                    'blindness', 'characterisation', 'colorimetry',
                    'constants', 'continuous', 'contrast', 'corresponding',
                    'difference', 'graph', 'io', 'models', 'notation',
                    'phenomena', 'plotting', 'quality', 'recovery',
                    'temperature', 'utilities', 'volume'):
    print(sub_package.title())
    pprint(getattr(colour, sub_package).__all__[:5] + ['...'])
    print('\n')
```

```
Adaptation
['CHROMATIC_ADAPTATION_TRANSFORMS',
 'XYZ_SCALING_CAT',
 'VON_KRIES_CAT',
 'BRADFORD_CAT',
 'SHARP_CAT',
 '...']


Algebra
['cartesian_to_spherical',
 'spherical_to_cartesian',
 'cartesian_to_polar',
 'polar_to_cartesian',
 'cartesian_to_cylindrical',
 '...']


Appearance
['Hunt_InductionFactors',
 'HUNT_VIEWING_CONDITIONS',
 'Hunt_Specification',
 'XYZ_to_Hunt',
 'ATD95_Specification',
 '...']


Biochemistry
['reaction_rate_MichealisMenten',
 'substrate_concentration_MichealisMenten',
 '...']


Blindness
```

(continues on next page)

(continued from previous page)

```
['CVD_MATRICES_MACHADO2010',
 'anomalous_trichromacy_cmfs_Machado2009',
 'anomalous_trichromacy_matrix_Machado2009',
 'cvd_matrix_Machado2009',
 '...']


Characterisation
['RGB_SpectralSensitivities',
 'RGB_DisplayPrimaries',
 'CAMERAS_RGB_SPECTRAL_SENSITIVITIES',
 'COLOURCHECKERS',
 'ColourChecker',
 '...']


Colorimetry
['SpectralShape',
 'DEFAULT_SPECTRAL_SHAPE',
 'SpectralDistribution',
 'MultiSpectralDistributions',
 'sds_and_multi_sds_to_sds',
 '...']


Constants
['K_M',
 'KP_M',
 'AVOGADRO_CONSTANT',
 'BOLTZMANN_CONSTANT',
 'LIGHT_SPEED',
 '...']


Continuous
['AbstractContinuousFunction', 'Signal', 'MultiSignals', '...']


Contrast
['optical_MTF_Barten1999',
 'pupil_diameter_Barten1999',
 'sigma_Barten1999',
 'retinal_illuminance_Barten1999',
 'maximum_angular_size_Barten1999',
 '...']


Corresponding
['BRENEMAN_EXPERIMENTS',
 'BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES',
 'CorrespondingColourDataset',
 'CorrespondingChromaticitiesPrediction',
 'corresponding_chromaticities_prediction_CIE1994',
 '...']


Difference
['delta_E_CAM02LCD',
 'delta_E_CAM02SCD',
 'delta_E_CAM02UCS',
 'delta_E_CAM16LCD',
```

(continues on next page)

(continued from previous page)

```
 'delta_E_CAM16SCD',
 '...']


Graph
['CONVERSION_GRAPH',
 'CONVERSION_GRAPH_NODE_LABELS',
 'describe_conversion_path',
 'convert',
 '...']


Io
['SpectralDistribution_IESTM2714',
 'AbstractLUTSequenceOperator',
 'LUT1D',
 'LUT3x1D',
 'LUT3D',
 '...']


Models
['JMh_CIECAM02_to_CAM02LCD',
 'CAM02LCD_to_JMh_CIECAM02',
 'JMh_CIECAM02_to_CAM02SCD',
 'CAM02SCD_to_JMh_CIECAM02',
 'JMh_CIECAM02_to_CAM02UCS',
 '...']


Notation
['MUNSELL_COLOURS_ALL',
 'MUNSELL_COLOURS_1929',
 'MUNSELL_COLOURS_REAL',
 'MUNSELL_COLOURS',
 'munsell_value',
 '...']


Phenomena
['scattering_cross_section',
 'rayleigh_optical_depth',
 'rayleigh_scattering',
 'sd_rayleigh_scattering',
 '...']


Plotting
['ASTMG173_ETR',
 'ASTMG173_GLOBAL_TILT',
 'ASTMG173_DIRECT_CIRCUMSOLAR',
 'COLOUR_STYLE_CONSTANTS',
 'COLOUR_ARROW_STYLE',
 '...']


Quality
['TCS_SDS',
 'VS_SDS',
 'CRI_Specification',
 'colour_rendering_index',
```

(continues on next page)

(continued from previous page)

```
 'CQS_Specification',
 '...']


Recovery
['SMITS_1999_SDS',
 'XYZ_to_sd_Meng2015',
 'RGB_to_sd_Smits1999',
 'XYZ_TO_SD_METHODS',
 'XYZ_to_sd',
 '...']


Temperature
['xy_to_CCT_CIE_D',
 'CCT_to_xy_CIE_D',
 'xy_to_CCT_Hernandez1999',
 'CCT_to_xy_Hernandez1999',
 'xy_to_CCT_Kang2002',
 '...']


Utilities
['Lookup',
 'Structure',
 'CaseInsensitiveMapping',
 'handle_numpy_errors',
 'ignore_numpy_errors',
 '...']


Volume
['ILLUMINANTS_OPTIMAL_COLOUR_STIMULI',
 'is_within_macadam_limits',
 'is_within_mesh_volume',
 'is_within_pointer_gamut',
 'generate_pulse_waves',
 '...']
```

The codebase is documented and most docstrings have usage examples:

```
print(colour.temperature.CCT_to_uv_Ohno2013.__doc__)
```

```
Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given
correlated colour temperature :math:`T_{cp}`, :math:`\Delta_{uv}` and
colour matching functions using *Ohno (2013)* method.

Parameters
----------
CCT_D_uv : ndarray
    Correlated colour temperature :math:`T_{cp}`, :math:`\Delta_{uv}`.
cmfs : XYZ_ColourMatchingFunctions, optional
    Standard observer colour matching functions.

Returns
-------
ndarray
    *CIE UCS* colourspace *uv* chromaticity coordinates.

References
----------
```

(continues on next page)

(continued from previous page)

```
:cite:`Ohno2014a`

Examples
--------
>>> from colour import DEFAULT_SPECTRAL_SHAPE, STANDARD_OBSERVERS_CMFS
>>> cmfs = (
...     STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer'].
...     copy().align(DEFAULT_SPECTRAL_SHAPE)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_uv_Ohno2013(CCT_D_uv, cmfs)  # doctest: +ELLIPSIS
array([ 0.1977999...,  0.3122004...])
```

At the core of **Colour** is the `colour.colorimetry` sub-package, it defines the objects needed for spectral computations and many others:

```
import colour.colorimetry as colorimetry

pprint(colorimetry.__all__)
```

```
['SpectralShape',
 'DEFAULT_SPECTRAL_SHAPE',
 'SpectralDistribution',
 'MultiSpectralDistributions',
 'sds_and_multi_sds_to_sds',
 'sd_blackbody',
 'blackbody_spectral_radiance',
 'planck_law',
 'LMS_ConeFundamentals',
 'RGB_ColourMatchingFunctions',
 'XYZ_ColourMatchingFunctions',
 'CMFS',
 'LMS_CMFS',
 'RGB_CMFS',
 'STANDARD_OBSERVERS_CMFS',
 'ILLUMINANTS',
 'D_ILLUMINANTS_S_SDS',
 'HUNTERLAB_ILLUMINANTS',
 'ILLUMINANTS_SDS',
 'LIGHT_SOURCES',
 'LIGHT_SOURCES_SDS',
 'LEFS',
 'PHOTOPIC_LEFS',
 'SCOTOPIC_LEFS',
 'sd_constant',
 'sd_zeros',
 'sd_ones',
 'SD_GAUSSIAN_METHODS',
 'sd_gaussian',
 'sd_gaussian_normal',
 'sd_gaussian_fwhm',
 'SD_SINGLE_LED_METHODS',
 'sd_single_led',
 'sd_single_led_Ohno2005',
 'SD_MULTI_LEDS_METHODS',
 'sd_multi_leds',
 'sd_multi_leds_Ohno2005',
 'SD_TO_XYZ_METHODS',
 'MULTI_SD_TO_XYZ_METHODS',
 'sd_to_XYZ',
 'multi_sds_to_XYZ',
```

(continues on next page)

(continued from previous page)

```
'ASTME308_PRACTISE_SHAPE',
'lagrange_coefficients_ASTME2022',
'tristimulus_weighting_factors_ASTME2022',
'adjust_tristimulus_weighting_factors_ASTME308',
'sd_to_XYZ_integration',
'sd_to_XYZ_tristimulus_weighting_factors_ASTME308',
'sd_to_XYZ_ASTME308',
'multi_sds_to_XYZ_integration',
'multi_sds_to_XYZ_ASTME308',
'wavelength_to_XYZ',
'BANDPASS_CORRECTION_METHODS',
'bandpass_correction',
'bandpass_correction_Stearns1988',
'sd_CIE_standard_illuminant_A',
'sd_CIE_illuminant_D_series',
'daylight_locus_function',
'sd_mesopic_luminous_efficiency_function',
'mesopic_weighting_function',
'LIGHTNESS_METHODS',
'lightness',
'lightness_Glasser1958',
'lightness_Wyszecki1963',
'lightness_CIE1976',
'lightness_Fairchild2010',
'lightness_Fairchild2011',
'intermediate_lightness_function_CIE1976',
'LUMINANCE_METHODS',
'luminance',
'luminance_Newhall1943',
'luminance_ASTMD1535',
'luminance_CIE1976',
'luminance_Fairchild2010',
'luminance_Fairchild2011',
'intermediate_luminance_function_CIE1976',
'dominant_wavelength',
'complementary_wavelength',
'excitation_purity',
'colorimetric_purity',
'luminous_flux',
'luminous_efficiency',
'luminous_efficacy',
'RGB_10_degree_cmfs_to_LMS_10_degree_cmfs',
'RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs',
'RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs',
'LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs',
'LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs',
'WHITENESS_METHODS',
'whiteness',
'whiteness_Berger1959',
'whiteness_Taube1960',
'whiteness_Stensby1968',
'whiteness_ASTME313',
'whiteness_Ganz1979',
'whiteness_CIE2004',
'YELLOWNESS_METHODS',
'yellowness',
'yellowness_ASTMD1925',
'yellowness_ASTME313']
```

Colour computations leverage a comprehensive quantity of datasets available in most sub-packages, for example the `colour.colorimetry.datasets` defines the following components:

```
import colour.colorimetry.datasets as datasets

pprint(datasets.__all__)
```

```
['CMFS',
 'LMS_CMFS',
 'RGB_CMFS',
 'STANDARD_OBSERVERS_CMFS',
 'ILLUMINANTS',
 'D_ILLUMINANTS_S_SDS',
 'HUNTERLAB_ILLUMINANTS',
 'ILLUMINANTS_SDS',
 'LIGHT_SOURCES',
 'LIGHT_SOURCES_SDS',
 'LEFS',
 'PHOTOPIC_LEFS',
 'SCOTOPIC_LEFS']
```

From Spectral Distribution

Whether it be a sample spectral distribution, colour matching functions or illuminants, spectral data is manipulated using an object built with the `colour.SpectralDistribution` class or based on it:

```
# Defining a sample spectral distribution data.
sample_sd_data = {
    380: 0.048,
    385: 0.051,
    390: 0.055,
    395: 0.060,
    400: 0.065,
    405: 0.068,
    410: 0.068,
    415: 0.067,
    420: 0.064,
    425: 0.062,
    430: 0.059,
    435: 0.057,
    440: 0.055,
    445: 0.054,
    450: 0.053,
    455: 0.053,
    460: 0.052,
    465: 0.052,
    470: 0.052,
    475: 0.053,
    480: 0.054,
    485: 0.055,
    490: 0.057,
    495: 0.059,
    500: 0.061,
    505: 0.062,
    510: 0.065,
    515: 0.067,
    520: 0.070,
    525: 0.072,
    530: 0.074,
    535: 0.075,
    540: 0.076,
    545: 0.078,
    550: 0.079,
```

(continues on next page)

(continued from previous page)

```
    555: 0.082,
    560: 0.087,
    565: 0.092,
    570: 0.100,
    575: 0.107,
    580: 0.115,
    585: 0.122,
    590: 0.129,
    595: 0.134,
    600: 0.138,
    605: 0.142,
    610: 0.146,
    615: 0.150,
    620: 0.154,
    625: 0.158,
    630: 0.163,
    635: 0.167,
    640: 0.173,
    645: 0.180,
    650: 0.188,
    655: 0.196,
    660: 0.204,
    665: 0.213,
    670: 0.222,
    675: 0.231,
    680: 0.242,
    685: 0.251,
    690: 0.261,
    695: 0.271,
    700: 0.282,
    705: 0.294,
    710: 0.305,
    715: 0.318,
    720: 0.334,
    725: 0.354,
    730: 0.372,
    735: 0.392,
    740: 0.409,
    745: 0.420,
    750: 0.436,
    755: 0.450,
    760: 0.462,
    765: 0.465,
    770: 0.448,
    775: 0.432,
    780: 0.421}

sd = colour.SpectralDistribution(sample_sd_data, name='Sample')
print(repr(sd))
```

```
SpectralDistribution([[  3.80000000e+02,   4.80000000e-02],
                      [  3.85000000e+02,   5.10000000e-02],
                      [  3.90000000e+02,   5.50000000e-02],
                      [  3.95000000e+02,   6.00000000e-02],
                      [  4.00000000e+02,   6.50000000e-02],
                      [  4.05000000e+02,   6.80000000e-02],
                      [  4.10000000e+02,   6.80000000e-02],
                      [  4.15000000e+02,   6.70000000e-02],
                      [  4.20000000e+02,   6.40000000e-02],
                      [  4.25000000e+02,   6.20000000e-02],
                      [  4.30000000e+02,   5.90000000e-02],
```

(continues on next page)

(continued from previous page)

```
                [  4.35000000e+02,   5.70000000e-02],
                [  4.40000000e+02,   5.50000000e-02],
                [  4.45000000e+02,   5.40000000e-02],
                [  4.50000000e+02,   5.30000000e-02],
                [  4.55000000e+02,   5.30000000e-02],
                [  4.60000000e+02,   5.20000000e-02],
                [  4.65000000e+02,   5.20000000e-02],
                [  4.70000000e+02,   5.20000000e-02],
                [  4.75000000e+02,   5.30000000e-02],
                [  4.80000000e+02,   5.40000000e-02],
                [  4.85000000e+02,   5.50000000e-02],
                [  4.90000000e+02,   5.70000000e-02],
                [  4.95000000e+02,   5.90000000e-02],
                [  5.00000000e+02,   6.10000000e-02],
                [  5.05000000e+02,   6.20000000e-02],
                [  5.10000000e+02,   6.50000000e-02],
                [  5.15000000e+02,   6.70000000e-02],
                [  5.20000000e+02,   7.00000000e-02],
                [  5.25000000e+02,   7.20000000e-02],
                [  5.30000000e+02,   7.40000000e-02],
                [  5.35000000e+02,   7.50000000e-02],
                [  5.40000000e+02,   7.60000000e-02],
                [  5.45000000e+02,   7.80000000e-02],
                [  5.50000000e+02,   7.90000000e-02],
                [  5.55000000e+02,   8.20000000e-02],
                [  5.60000000e+02,   8.70000000e-02],
                [  5.65000000e+02,   9.20000000e-02],
                [  5.70000000e+02,   1.00000000e-01],
                [  5.75000000e+02,   1.07000000e-01],
                [  5.80000000e+02,   1.15000000e-01],
                [  5.85000000e+02,   1.22000000e-01],
                [  5.90000000e+02,   1.29000000e-01],
                [  5.95000000e+02,   1.34000000e-01],
                [  6.00000000e+02,   1.38000000e-01],
                [  6.05000000e+02,   1.42000000e-01],
                [  6.10000000e+02,   1.46000000e-01],
                [  6.15000000e+02,   1.50000000e-01],
                [  6.20000000e+02,   1.54000000e-01],
                [  6.25000000e+02,   1.58000000e-01],
                [  6.30000000e+02,   1.63000000e-01],
                [  6.35000000e+02,   1.67000000e-01],
                [  6.40000000e+02,   1.73000000e-01],
                [  6.45000000e+02,   1.80000000e-01],
                [  6.50000000e+02,   1.88000000e-01],
                [  6.55000000e+02,   1.96000000e-01],
                [  6.60000000e+02,   2.04000000e-01],
                [  6.65000000e+02,   2.13000000e-01],
                [  6.70000000e+02,   2.22000000e-01],
                [  6.75000000e+02,   2.31000000e-01],
                [  6.80000000e+02,   2.42000000e-01],
                [  6.85000000e+02,   2.51000000e-01],
                [  6.90000000e+02,   2.61000000e-01],
                [  6.95000000e+02,   2.71000000e-01],
                [  7.00000000e+02,   2.82000000e-01],
                [  7.05000000e+02,   2.94000000e-01],
                [  7.10000000e+02,   3.05000000e-01],
                [  7.15000000e+02,   3.18000000e-01],
                [  7.20000000e+02,   3.34000000e-01],
                [  7.25000000e+02,   3.54000000e-01],
                [  7.30000000e+02,   3.72000000e-01],
                [  7.35000000e+02,   3.92000000e-01],
```

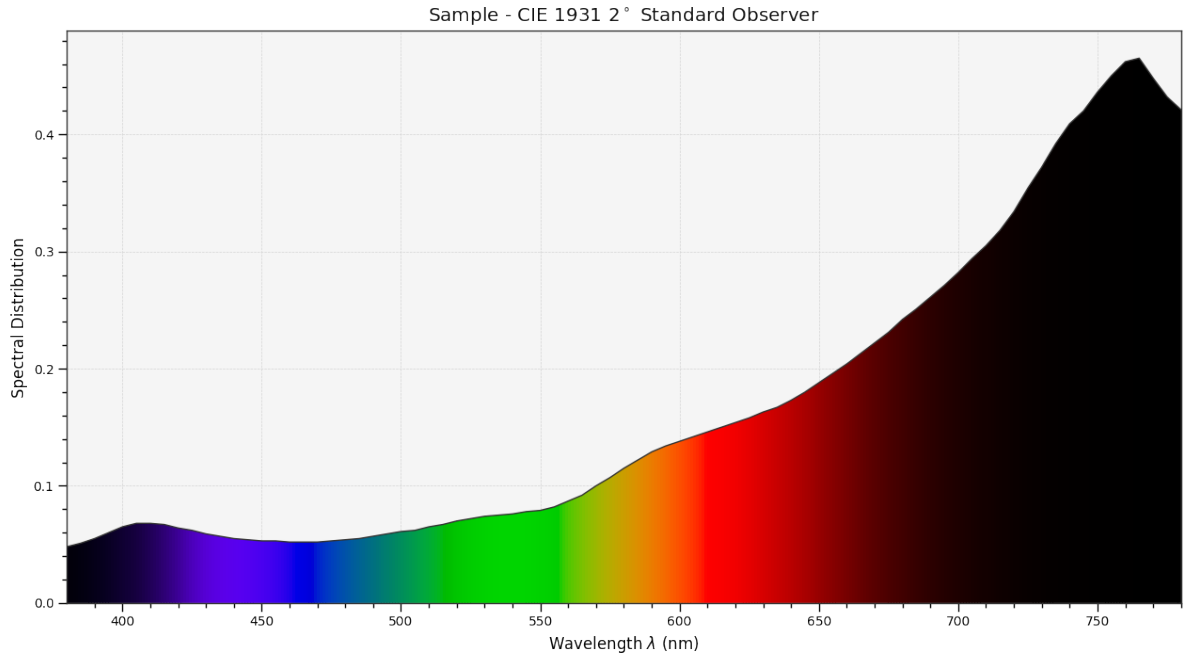
(continues on next page)

(continued from previous page)

```
                    [  7.40000000e+02,   4.09000000e-01],
                    [  7.45000000e+02,   4.20000000e-01],
                    [  7.50000000e+02,   4.36000000e-01],
                    [  7.55000000e+02,   4.50000000e-01],
                    [  7.60000000e+02,   4.62000000e-01],
                    [  7.65000000e+02,   4.65000000e-01],
                    [  7.70000000e+02,   4.48000000e-01],
                    [  7.75000000e+02,   4.32000000e-01],
                    [  7.80000000e+02,   4.21000000e-01]],
                   interpolator=SpragueInterpolator,
                   interpolator_args={},
                   extrapolator=Extrapolator,
                   extrapolator_args={u'right': None, u'method': u'Constant', u'left': None})
```

The sample spectral distribution can be easily plotted against the visible spectrum:

```
# Plotting the sample spectral distribution.
plot_single_sd(sd)
```



With the sample spectral distribution defined, its shape is retrieved as follows:

```
# Displaying the sample spectral distribution shape.
print(sd.shape)
```

```
(380.0, 780.0, 5.0)
```

The returned shape is an instance of the `colour.SpectralShape` class:

```
repr(sd.shape)
```

```
'SpectralShape(380.0, 780.0, 5.0)'
```

The `colour.SpectralShape` class is used throughout **Colour** to define spectral dimensions and is instantiated as follows:

```
# Using *colour.SpectralShape* with iteration.
shape = colour.SpectralShape(start=0, end=10, interval=1)
for wavelength in shape:
    print(wavelength)

# *colour.SpectralShape.range* method is providing the complete range of values.
shape = colour.SpectralShape(0, 10, 0.5)
shape.range()
```

```
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
```

```
array([  0. ,   0.5,   1. ,   1.5,   2. ,   2.5,   3. ,   3.5,   4. ,
         4.5,   5. ,   5.5,   6. ,   6.5,   7. ,   7.5,   8. ,   8.5,
         9. ,   9.5,  10. ])
```

Colour defines three convenient objects to create constant spectral distributions:

- `colour.sd_constant`
- `colour.sd_zeros`
- `colour.sd_ones`

```
# Defining a constant spectral distribution.
sd_constant = colour.sd_constant(100)
print('"Constant Spectral Distribution"')
print(sd_constant.shape)
print(sd_constant[400])

# Defining a zeros filled spectral distribution.
print('\n"Zeros Filled Spectral Distribution"')
sd_zeros = colour.sd_zeros()
print(sd_zeros.shape)
print(sd_zeros[400])

# Defining a ones filled spectral distribution.
print('\n"Ones Filled Spectral Distribution"')
sd_ones = colour.sd_ones()
print(sd_ones.shape)
print(sd_ones[400])
```

```
"Constant Spectral Distribution"
(360.0, 780.0, 1.0)
100.0

"Zeros Filled Spectral Distribution"
(360.0, 780.0, 1.0)
0.0

"Ones Filled Spectral Distribution"
(360.0, 780.0, 1.0)
1.0
```

By default the shape used by `colour.sd_constant`, `colour.sd_zeros` and `colour.sd_ones` is the one defined by the `colour.DEFAULT_SPECTRAL_SHAPE` attribute and based on *ASTM E308-15* practise shape.

```
print(repr(colour.DEFAULT_SPECTRAL_SHAPE))
```

```
SpectralShape(360, 780, 1)
```

A custom shape can be passed to construct a constant spectral distribution with user defined dimensions:

```
colour.sd_ones(colour.SpectralShape(400, 700, 5))[450]
```

```
1.0
```

The `colour.SpectralDistribution` class supports the following arithmetical operations:

- *addition*
- *subtraction*
- *multiplication*
- *division*
- *exponentiation*

```
sd1 = colour.sd_ones()
print('"Ones Filled Spectral Distribution"')
print(sd1[400])

print('\n"x2 Constant Multiplied"')
print((sd1 * 2)[400])

print('\n"+ Spectral Distribution"')
print((sd1 + colour.sd_ones())[400])
```

```
"Ones Filled Spectral Distribution"
1.0

"x2 Constant Multiplied"
2.0

"+ Spectral Distribution"
2.0
```

Often interpolation of the spectral distribution is required, this is achieved with the `colour.SpectralDistribution.interpolate` method. Depending on the wavelengths uniformity, the default interpolation method will differ. Following *CIE 167:2005* recommendation: The method developed by *Sprague (1880)* should be used for interpolating functions having a uniformly spaced independent variable and a *Cubic Spline* method for non-uniformly spaced independent variable [CIET13805a].

The uniformity of the sample spectral distribution is assessed as follows:

```
# Checking the sample spectral distribution uniformity.
print(sd.is_uniform())
```

```
True
```

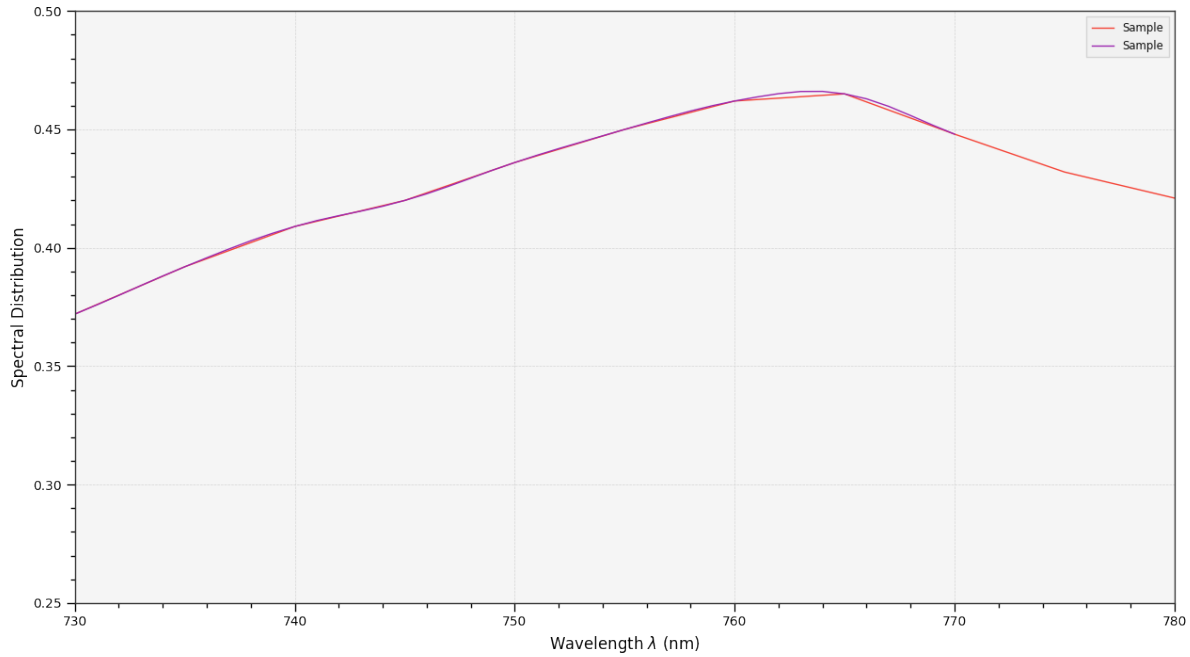
In this case, since the sample spectral distribution is uniform the interpolation defaults to the `colour.SpragueInterpolator` interpolator.

Note: Interpolation happens in place and may alter the original data, use the `colour.SpectralDistribution.copy` method to generate a copy of the spectral distribution before interpolation.

```
# Copying the sample spectral distribution.
sd_copy = sd.copy()

# Interpolating the copied sample spectral distribution.
sd_copy.interpolate(colour.SpectralShape(400, 770, 1))
sd_copy[401]
```

```
0.065809599999999996
```

```
# Comparing the interpolated spectral distribution with the original one.
plot_multi_sds([sd, sd_copy], bounding_box=[730,780, 0.25, 0.5])
```



Extrapolation although dangerous can be used to help aligning two spectral distributions together. *CIE publication CIE 15:2004 "Colorimetry"* recommends that unmeasured values may be set equal to the nearest measured value of the appropriate quantity in truncation [CIET14804d]:

```
# Extrapolating the copied sample spectral distribution.
sd_copy.extrapolate(colour.SpectralShape(340, 830))
sd_copy[340], sd_copy[830]
```

```
(0.065000000000000002, 0.44800000000000018)
```

The underlying interpolator can be swapped for any of the **Colour** interpolators:

```
pprint([
    export for export in colour.algebra.interpolation.__all__
    if 'Interpolator' in export
])
```

```
[u'KernelInterpolator',
 u'LinearInterpolator',
 u'SpragueInterpolator',
 u'CubicSplineInterpolator',
 u'PchipInterpolator',
 u'NullInterpolator']
```

```
# Changing interpolator while trimming the copied spectral distribution.
sd_copy.interpolate(
    colour.SpectralShape(400, 700, 10), interpolator=colour.LinearInterpolator)
```

```
SpectralDistribution([[  4.00000000e+02,   6.50000000e-02],
                      [  4.10000000e+02,   6.80000000e-02],
                      [  4.20000000e+02,   6.40000000e-02],
                      [  4.30000000e+02,   5.90000000e-02],
                      [  4.40000000e+02,   5.50000000e-02],
                      [  4.50000000e+02,   5.30000000e-02],
                      [  4.60000000e+02,   5.20000000e-02],
                      [  4.70000000e+02,   5.20000000e-02],
                      [  4.80000000e+02,   5.40000000e-02],
                      [  4.90000000e+02,   5.70000000e-02],
                      [  5.00000000e+02,   6.10000000e-02],
                      [  5.10000000e+02,   6.50000000e-02],
                      [  5.20000000e+02,   7.00000000e-02],
                      [  5.30000000e+02,   7.40000000e-02],
                      [  5.40000000e+02,   7.60000000e-02],
                      [  5.50000000e+02,   7.90000000e-02],
                      [  5.60000000e+02,   8.70000000e-02],
                      [  5.70000000e+02,   1.00000000e-01],
                      [  5.80000000e+02,   1.15000000e-01],
                      [  5.90000000e+02,   1.29000000e-01],
                      [  6.00000000e+02,   1.38000000e-01],
                      [  6.10000000e+02,   1.46000000e-01],
                      [  6.20000000e+02,   1.54000000e-01],
                      [  6.30000000e+02,   1.63000000e-01],
                      [  6.40000000e+02,   1.73000000e-01],
                      [  6.50000000e+02,   1.88000000e-01],
                      [  6.60000000e+02,   2.04000000e-01],
                      [  6.70000000e+02,   2.22000000e-01],
                      [  6.80000000e+02,   2.42000000e-01],
                      [  6.90000000e+02,   2.61000000e-01],
                      [  7.00000000e+02,   2.82000000e-01]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={u'right': None, u'method': u'Constant', u'left': None})
```

The extrapolation behaviour can be changed for `Linear` method instead of the `Constant` default method or even use arbitrary constant `left` and `right` values:

```
# Extrapolating the copied sample spectral distribution with *Linear* method.
sd_copy.extrapolate(
    colour.SpectralShape(340, 830),
    extrapolator_args={'method': 'Linear',
                       'right': 0})
sd_copy[340], sd_copy[830]
```

```
(0.046999999999999348, 0.0)
```

Aligning a spectral distribution is a convenient way to first interpolates the current data within its original bounds, then, if required, extrapolate any missing values to match the requested shape:

```
# Aligning the cloned sample spectral distribution.
# The spectral distribution is first trimmed as above.
sd_copy.interpolate(colour.SpectralShape(400, 700))
sd_copy.align(colour.SpectralShape(340, 830, 5))
sd_copy[340], sd_copy[830]
```

```
(0.065000000000000002, 0.28199999999999975)
```

The `colour.SpectralDistribution` class also supports various arithmetic operations like *addition*, *subtraction*, *multiplication*, *division* or *exponentiation* with *numeric* and *array_like* variables or other `colour.SpectralDistribution` class instances:

```
sd = colour.SpectralDistribution({
    410: 0.25,
    420: 0.50,
    430: 0.75,
    440: 1.0,
    450: 0.75,
    460: 0.50,
    480: 0.25
})

print((sd.copy() + 1).values)
print((sd.copy() * 2).values)
print((sd * [0.35, 1.55, 0.75, 2.55, 0.95, 0.65, 0.15]).values)
print((sd * colour.sd_constant(2, sd.shape) * colour.sd_constant(3, sd.shape)).values)
```

```
[ 1.25  1.5   1.75  2.    1.75  1.5   1.25]
[ 0.5  1.   1.5  2.   1.5  1.   0.5]
[ 0.0875  0.775   0.5625  2.55    0.7125  0.325   0.0375]
[ 1.5  3.   4.5  6.   4.5  3.   nan  1.5]
```

The spectral distribution can be normalised with an arbitrary factor:

```
print(sd.normalise().values)
print(sd.normalise(100).values)
```

```
[ 0.25  0.5   0.75  1.    0.75  0.5   0.25]
[  25.   50.   75.  100.   75.   50.   25.]
```

A the heart of the `colour.SpectralDistribution` class is the `colour.continuous.Signal` class which implements the `colour.continuous.Signal.function` method.

Evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable.

It adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the `colour.continuous.Signal.domain` attribute corresponds with the function dependent and already known range stored in the `colour.continuous.Signal.range` attribute.

Describing the `colour.continuous.Signal` class is beyond the scope of this tutorial but the core capability can be described.

```
import numpy as np

range_ = np.linspace(10, 100, 10)
signal = colour.continuous.Signal(range_)
print(repr(signal))
```

```
Signal([[   0.,   10.],
        [   1.,   20.],
        [   2.,   30.],
        [   3.,   40.],
        [   4.,   50.],
        [   5.,   60.],
        [   6.,   70.],
```

(continues on next page)

(continued from previous page)

```
        [   7.,   80.],
        [   8.,   90.],
        [   9.,  100.]],
       interpolator=KernelInterpolator,
       interpolator_args={},
       extrapolator=Extrapolator,
       extrapolator_args={u'right': nan, u'method': u'Constant', u'left': nan})
```

```
# Returning the corresponding range *y* variable for any arbitrary independent domain *x* variable.
signal[np.random.uniform(0, 9, 10)]
```

```
array([ 55.91309735,  65.4172615 ,  65.54495059,  88.17819416,
        61.88860248,  10.53878826,  55.25130534,  46.14659783,
        86.41406136,  84.59897703])
```

Convert to Tristimulus Values

From a given spectral distribution, *CIE XYZ* tristimulus values can be calculated:

```
sd = colour.SpectralDistribution(sample_sd_data)
cmfs = colour.STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
illuminant = colour.ILLUMINANTS_SDS['D65']

# Calculating the sample spectral distribution *CIE XYZ* tristimulus values.
XYZ = colour.sd_to_XYZ(sd, cmfs, illuminant)
print(XYZ)
```

```
[ 10.97085572   9.70278591   6.05562778]
```

From *CIE XYZ* Colourspace

CIE XYZ is the central colourspace for Colour Science from which many computations are available, expanding to even more computations:

```
# Displaying objects interacting directly with the *CIE XYZ* colourspace.
pprint([name for name in colour.__all__ if name.startswith('XYZ_to')])
```

```
['XYZ_to_ATD95',
 'XYZ_to_CAM16',
 'XYZ_to_CIECAM02',
 'XYZ_to_Hunt',
 'XYZ_to_LLAB',
 'XYZ_to_Nayatani95',
 'XYZ_to_RLAB',
 'XYZ_to_Hunter_Lab',
 'XYZ_to_Hunter_Rdab',
 'XYZ_to_IPT',
 'XYZ_to_JzAzBz',
 'XYZ_to_K_ab_HunterLab1966',
 'XYZ_to_Lab',
 'XYZ_to_Luv',
 'XYZ_to_OSA_UCS',
 'XYZ_to_RGB',
 'XYZ_to_UCS',
 'XYZ_to_UVW',
 'XYZ_to_hdr_CIELab',
```

(continues on next page)

(continued from previous page)

```
 'XYZ_to_hdr_IPT',
 'XYZ_to_sRGB',
 'XYZ_to_xy',
 'XYZ_to_xyY',
 'XYZ_to_sd']
```

Convert to Display Colours

CIE XYZ tristimulus values can be converted into *sRGB* colourspace *RGB* values in order to display them on screen:

```
# The output domain of *colour.sd_to_XYZ* is [0, 100] and the input
# domain of *colour.XYZ_to_sRGB* is [0, 1]. It needs to be accounted for,
# thus the input *CIE XYZ* tristimulus values are scaled.
RGB = colour.XYZ_to_sRGB(XYZ / 100)
print(RGB)
```

```
[ 0.45675795  0.30986982  0.24861924]
```

```
# Plotting the *sRGB* colourspace colour of the *Sample* spectral distribution.
plot_single_colour_swatch(
    ColourSwatch('Sample', RGB),
    text_parameters={'size': 'x-large'})
```

Generate Colour Rendition Charts

Likewise, colour values from a colour rendition chart sample can be computed.

Note: This is useful for render time checks in the VFX industry, where a synthetic colour chart can be inserted into a render to ensure the colour management is acting as expected.

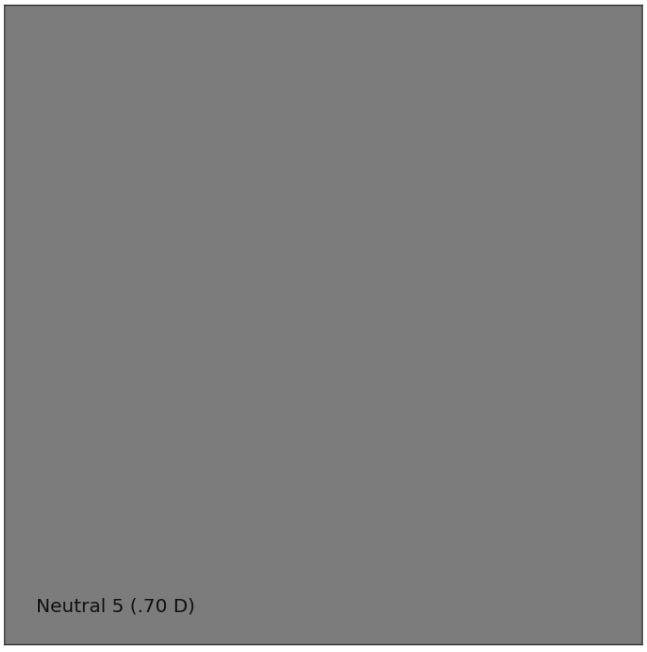
The `colour.characterisation` sub-package contains the dataset for various colour rendition charts:

```
# Colour rendition charts chromaticity coordinates.
print(sorted(colour.characterisation.COLOURCHECKERS.keys()))

# Colour rendition charts spectral distributions.
print(sorted(colour.characterisation.COLOURCHECKERS_SDS.keys()))
```

```
[u'BabelColor Average', u'ColorChecker 1976', u'ColorChecker 2005', u'ColorChecker24 - After␣
↪November 2014', u'ColorChecker24 - Before November 2014', u'babel_average', u'cc2005', u'cca2014',
↪ u'ccb2014']
[u'BabelColor Average', u'ColorChecker N Ohta', u'babel_average', u'cc_ohta']
```

Note: The above `cc2005`, `babel_average` and `cc_ohta` keys are convenient aliases for respectively `ColorChecker 2005`, `BabelColor Average` and `ColorChecker N Ohta` keys.

```
# Plotting the *sRGB* colourspace colour of *neutral 5 (.70 D)* patch.
patch_name = 'neutral 5 (.70 D)'
patch_sd = colour.COLOURCHECKERS_SDS['ColorChecker N Ohta'][patch_name]
XYZ = colour.sd_to_XYZ(patch_sd, cmfs, illuminant)
RGB = colour.XYZ_to_sRGB(XYZ / 100)

plot_single_colour_swatch(
    ColourSwatch(patch_name.title(), RGB),
    text_parameters={'size': 'x-large'})
```



Colour defines a convenient plotting object to draw synthetic colour rendition charts figures:

```
plot_single_colour_checker(
    colour_checker='ColorChecker 2005', text_parameters={'visible': False})
```

Convert to Chromaticity Coordinates

Given a spectral distribution, chromaticity coordinates *CIE xy* can be computed using the `colour.XYZ_to_xy` definition:

```
# Computing *CIE xy* chromaticity coordinates for the *neutral 5 (.70 D)* patch.
xy =  colour.XYZ_to_xy(XYZ)
print(xy)
```

```
[ 0.31259787  0.32870029]
```

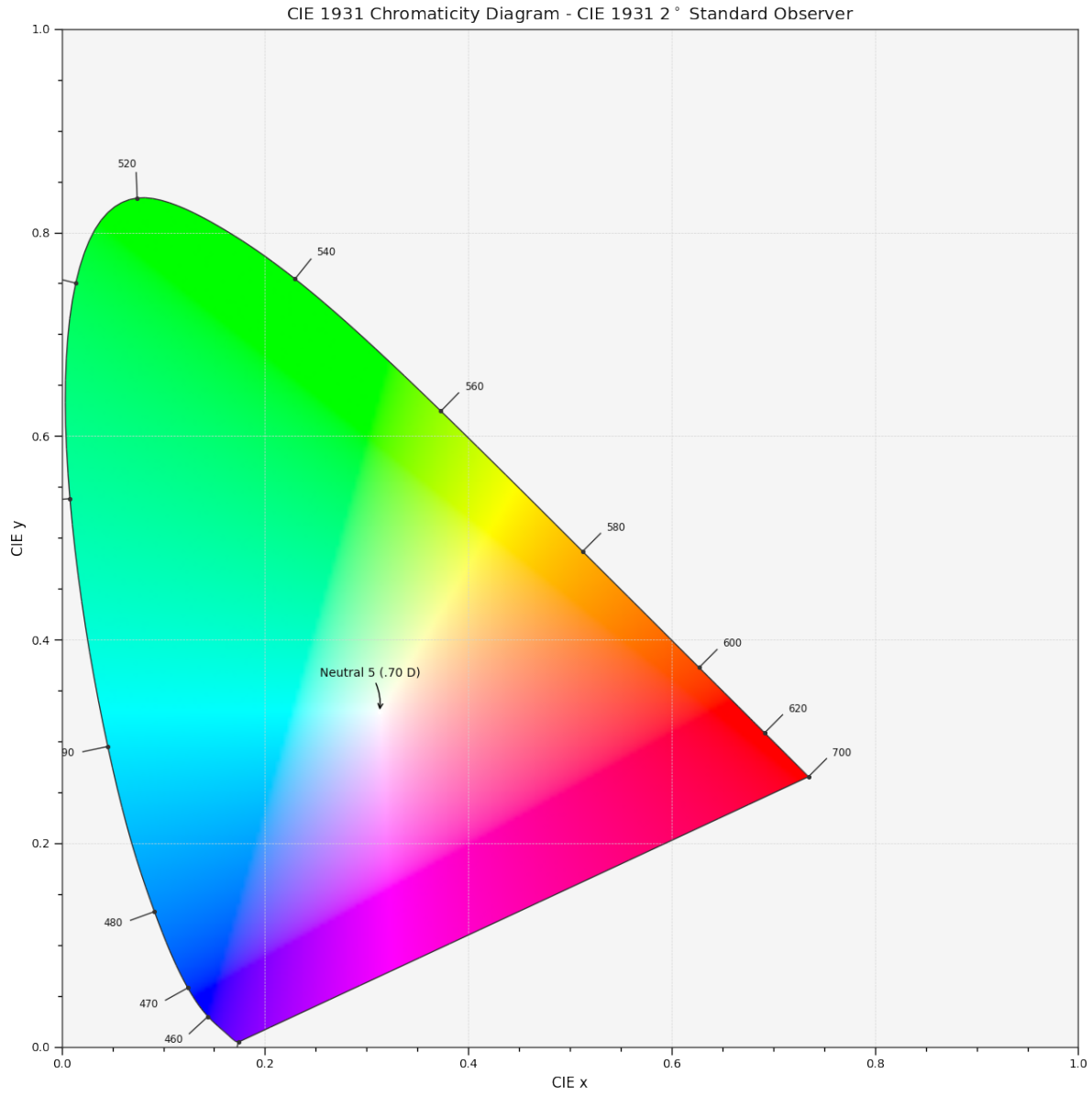
Chromaticity coordinates *CIE xy* can be plotted into the *CIE 1931 Chromaticity Diagram*:

```
import matplotlib.pyplot as plt

# Plotting the *CIE 1931 Chromaticity Diagram*.
# The argument *standalone=False* is passed so that the plot doesn't get
# displayed and can be used as a basis for other plots.
plot_chromaticity_diagram_CIE1931(standalone=False)

# Plotting the *CIE xy* chromaticity coordinates.
x, y = xy
plt.plot(x, y, 'o-', color='white')

# Annotating the plot.
plt.annotate(patch_sd.name.title(),
             xy=xy,
             xytext=(-50, 30),
             textcoords='offset points',
             arrowprops=dict(arrowstyle='->', connectionstyle='arc3, rad=-0.2'))

# Displaying the plot.
render(
```

(continues on next page)

(continued from previous page)

```
    standalone=True,
    limits=(-0.1, 0.9, -0.1, 0.9),
    x_tighten=True,
    y_tighten=True)
```



And More...

With the hope that this small introduction was useful and gave envy to see more, a good place to explore the API further is the Jupyter Notebooks page.

4.3.1.2 Basics

N-Dimensional Arrays Support

Most of Colour definitions are fully vectorised and support n-dimensional arrays by leveraging Numpy.

While it is recommended to use ndarrays as input for the API objects, it is possible to use tuples or lists:

```
import colour

xyY = (0.4316, 0.3777, 0.1008)
colour.xyY_to_XYZ(xyY)
```

```
array([ 0.11518475,  0.1008    ,  0.05089373])
```

```
xyY = [0.4316, 0.3777, 0.1008]
colour.xyY_to_XYZ(xyY)
```

```
array([ 0.11518475,  0.1008    ,  0.05089373])
```

```
xyY = [
    (0.4316, 0.3777, 0.1008),
    (0.4316, 0.3777, 0.1008),
    (0.4316, 0.3777, 0.1008),
]
colour.xyY_to_XYZ(xyY)
```

```
array([[ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373]])
```

As shown in the above example, there is widespread support for n-dimensional arrays:

```
import numpy as np

xyY = np.array([0.4316, 0.3777, 0.1008])
xyY = np.tile(xyY, (6, 1))
colour.xyY_to_XYZ(xyY)
```

```
array([[ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373],
       [ 0.11518475,  0.1008    ,  0.05089373]])
```

```
colour.xyY_to_XYZ(xyY.reshape([2, 3, 3]))
```

```
array([[[ 0.11518475,  0.1008    ,  0.05089373],
        [ 0.11518475,  0.1008    ,  0.05089373],
        [ 0.11518475,  0.1008    ,  0.05089373]],
```

(continues on next page)

(continued from previous page)

```
       [[ 0.11518475,  0.1008    ,  0.05089373],
        [ 0.11518475,  0.1008    ,  0.05089373],
        [ 0.11518475,  0.1008    ,  0.05089373]])
```

Which enables image processing:

```
RGB = colour.read_image('_static/Logo_Small_001.png')
RGB = RGB[..., 0:3]  # Discarding alpha channel.
XYZ = colour.sRGB_to_XYZ(RGB)
colour.plotting.plot_image(XYZ, text_parameters={'text': 'sRGB to XYZ'})
```

Domain-Range Scales

Note: This section has important information.

Colour adopts 4 main input domains and output ranges:

- *Scalars* usually in domain-range *[0, 1]* (or *[0, 10]* for *Munsell Value*).
- *Percentages* usually in domain-range *[0, 100]*.
- *Degrees* usually in domain-range *[0, 360]*.
- *Integers* usually in domain-range *[0, 2**n -1]* where *n* is the bit depth.

It is error prone but it is also a direct consequence of the inconsistency of the colour science field itself. We have discussed at length about this and we were leaning toward normalisation of the whole API to domain-range *[0, 1]*, we never committed for reasons highlighted by the following points:

- Colour Scientist performing computations related to Munsell Renotation System would be very surprised if the output *Munsell Value* was in range *[0, 1]* or *[0, 100]*.
- A Visual Effect Industry artist would be astonished to find out that conversion from *CIE XYZ* to *sRGB* was yielding values in range *[0, 100]*.

However benefits of having a consistent and predictable domain-range scale are numerous thus with Colour 0.3.12 we have introduced a mechanism to allow users to work within one of the two available domain-range scales.

Scale - Reference

'Reference' is the default domain-range scale of **Colour**, objects adopt the implemented reference, i.e. paper, publication, etc.., domain-range scale.

The **'Reference'** domain-range scale is inconsistent, e.g. colour appearance models, spectral conversions are typically in domain-range *[0, 100]* while RGB models will operate in domain-range *[0, 1]*. Some objects, e.g. `colour.colorimetry.lightness_Fairchild2011()` definition have mismatched domain-range: input domain *[0, 1]* and output range *[0, 100]*.

Scale - 1

'1' is a domain-range scale converting all the relevant objects from **Colour** public API to domain-range *[0, 1]*:

- *Scalars* in domain-range *[0, 10]*, e.g *Munsell Value* are scaled by *10*.
- *Percentages* in domain-range *[0, 100]* are scaled by *100*.
- *Degrees* in domain-range *[0, 360]* are scaled by *360*.
- *Integers* in domain-range *[0, 2**n -1]* where *n* is the bit depth are scaled by *2**n -1*.

Warning: The conversion to **'1'** domain-range scale is a *soft* normalisation and similarly to the **'Reference'** domain-range scale it is normal to encounter values exceeding *1*, e.g. High Dynamic Range Imagery (HDRI) or negative values, e.g. out-of-gamut RGB colourspace values. Some definitions such as `colour.models.eotf_ST2084()` which decodes absolute luminance values are not affected by any domain-range scales.

Understanding the Domain-Range Scale of an Object

Using `colour.adaptation.chromatic_adaptation_CIE1994()` definition docstring as an example, the *Notes* section features two tables.

The first table is for the domain, and lists the input arguments affected by the two domain-range scales and which normalisation they should adopt depending the domain-range scale in use:

Domain	Scale - Reference	Scale - 1
XYZ_1	[0, 100]	[0, 1]
Y_o	[0, 100]	[0, 1]

The second table is for the range and lists the return value of the definition:

Range	Scale - Reference	Scale - 1
XYZ_2	[0, 100]	[0, 1]

Working with the Domain-Range Scales

The current domain-range scale is returned with the `colour.get_domain_range_scale()` definition:

```
import colour

colour.get_domain_range_scale()
```

```
u'reference'
```

Changing from the **'Reference'** default domain-range scale to **'1'** is done with the `colour.set_domain_range_scale()` definition:

```
XYZ_1 = [28.00, 21.26, 5.27]
xy_o1 = [0.4476, 0.4074]
xy_o2 = [0.3127, 0.3290]
Y_o = 20
E_o1 = 1000
E_o2 = 1000
colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
array([ 24.03379521,  21.15621214,  17.64301199])
```

```
colour.set_domain_range_scale('1')

XYZ_1 = [0.2800, 0.2126, 0.0527]
Y_o = 0.2
colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
array([ 0.24033795,  0.21156212,  0.17643012])
```

The output tristimulus values with the **'1'** domain-range scale are equal to those from **'Reference'** default domain-range scale divided by *100*.

Passing incorrectly scaled values to the `colour.adaptation.chromatic_adaptation_CIE1994()` definition would result in unexpected values and a warning in that case:

```
colour.set_domain_range_scale('Reference')

colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
File "<ipython-input-...>", line 4, in <module>
  E_o2)
File "/colour-science/colour/colour/adaptation/cie1994.py", line 134, in chromatic_adaptation_
↪CIE1994
  warning(('"Y_o" luminance factor must be in [18, 100] domain, '
/colour-science/colour/colour/utilities/verbose.py:207: ColourWarning: "Y_o" luminance factor must␣
↪be in [18, 100] domain, unpredictable results may occur!
  warn(*args, **kwargs)
array([ 0.17171825,  0.13731098,  0.09972054])
```

Setting the **'1'** domain-range scale has the following effect on the `colour.adaptation.chromatic_adaptation_CIE1994()` definition:

As it expects values in domain *[0, 100]*, scaling occurs and the relevant input values, i.e. the values listed in the domain table, `XYZ_1` and `Y_o` are converted from domain *[0, 1]* to domain *[0, 100]* by `colour.utilities.to_domain_100()` definition and conversely return value `XYZ_2` is converted from range *[0, 100]* to range *[0, 1]* by `colour.utilities.from_range_100()` definition.

A convenient alternative to the `colour.set_domain_range_scale()` definition is the `colour.domain_range_scale` context manager and decorator. It temporarily overrides **Colour** domain-range

scale with given scale value:

```
with colour.domain_range_scale('1'):
    colour.adaptation.chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
```

```
[ 0.24033795  0.21156212  0.17643012]
```

Multiprocessing on Windows with Domain-Range Scales

Windows does not have a fork system call, a consequence is that child processes do not necessarily inherit from changes made to global variables.

It has crucial consequences as **Colour** stores the current domain-range scale into a global variable.

The solution is to define an initialisation definition that defines the scale upon child processes spawning.

The `colour.utilities.multiprocessing_pool` context manager conveniently performs the required initialisation so that the domain-range scale is propagated appropriately to child processes.

4.3.1.3 Advanced

Environment

Various environment variables can be used to modify **Colour** behaviour at runtime:

- *COLOUR_SCIENCE__COLOUR__SHOW_WARNINGS_WITH_TRACEBACK*: result in the `warnings.showwarning()` definition to be replaced with the `colour.utilities.show_warning()` definition and thus providing complete traceback from the point where the warning occurred.

Using Colour without Scipy

With the release of Colour 0.3.8, SciPy became a requirement.

Scipy is notoriously hard to compile, especially on Windows. Some Digital Content Creation (DCC) applications are shipping Python interpreters compiled with versions of Visual Studio such as 2011 or 2015. Those are incompatible with the Python Wheels commonly built with Visual Studio 2008 (Python 2.7) or Visual Studio 2017 (Python 3.6).

It is however possible to use **Colour** in a partially broken and mock **Scipy** by using the mock_for_colour.py module.

Assuming it is available for import, a typical usage would be as follows:

```
import sys
from mock_for_colour import MockModule

for module in ('scipy', 'scipy.interpolate', 'scipy.spatial',
               'scipy.spatial.distance', 'scipy.optimize'):
    sys.modules[str(module)] = MockModule(str(module))

import colour

xyY = (0.4316, 0.3777, 0.1008)
colour.xyY_to_XYZ(xyY)
```

```
array([ 0.11518475,  0.1008    ,  0.05089373])
```

Or directly using the `mock_scipy_for_colour` definition:

```
from mock_for_colour import mock_scipy_for_colour

mock_scipy_for_colour()

import colour

xyY = (0.4316, 0.3777, 0.1008)
colour.xyY_to_XYZ(xyY)
```

```
array([ 0.11518475,  0.1008    ,  0.05089373])
```

Anything relying on the spectral code will be unusable, but a great amount of useful functionality will still be available.

4.3.1.4 Reference

Colour

Chromatic Adaptation

- *Chromatic Adaptation*
- *Fairchild (1990)*
- *CIE 1994*
- *CMCCAT2000*
- *Von Kries*

Chromatic Adaptation

colour

chromatic_adaptation(XYZ, XYZ_w, XYZ_wr[, ...])	Adapts given stimulus from test viewing conditions to reference viewing conditions.
CHROMATIC_ADAPTATION_METHODS	Supported chromatic adaptation methods.
CMCCAT2000_VIEWING_CONDITIONS	Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

colour.chromatic_adaptation

colour.**chromatic_adaptation**(*XYZ*, *XYZ_w*, *XYZ_wr*, *method='Von Kries'*, ***kwargs*)

Adapts given stimulus from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.

- **method** (unicode, optional) – **{'Von Kries', 'CIE 1994', 'CMCCAT2000', 'Fairchild 1990'}**, Computation method.

Other Parameters

- **E_o1** (*numeric*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Test illuminance E_{o1} in cd/m^2.
- **E_o2** (*numeric*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Reference illuminance E_{o2} in cd/m^2.
- **L_A1** (*numeric or array_like*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (*numeric or array_like*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, Luminance of reference adapting field L_{A2} in cd/m^2.
- **Y_n** (*numeric or array_like*) – {colour.adaptation.chromatic_adaptation_Fairchild1990()}, Luminance Y_n of test adapting stimulus in cd/m^2.
- **Y_o** (*numeric*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Luminance factor Y_o of achromatic background normalised to domain [0.18, 1] in **'Reference'** domain-range scale.
- **direction** (*unicode, optional*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, **{'Forward', 'Inverse'}**, Chromatic adaptation direction.
- **discount_illuminant** (*bool, optional*) – {colour.adaptation.chromatic_adaptation_Fairchild1990()}, Truth value indicating if the illuminant should be discounted.
- **n** (*numeric, optional*) – {colour.adaptation.chromatic_adaptation_CIE1994()}, Noise component in fundamental primary system.
- **surround** (*CMCCAT2000_InductionFactors, optional*) – {colour.adaptation.chromatic_adaptation_CMCCAT2000()}, Surround viewing conditions induction factors.
- **transform** (*unicode, optional*) – {colour.adaptation.chromatic_adaptation_VonKries()}, **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns *CIE XYZ_c* tristimulus values of the stimulus corresponding colour.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
XYZ_w	[0, 1]	[0, 1]
XYZ_wr	[0, 1]	[0, 1]
Y_o	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_c	[0, 1]	[0, 1]

References

[CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

Examples

Von Kries chromatic adaptation:

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation(XYZ, XYZ_w, XYZ_wr)
...
array([ 0.2163881...,  0.1257    ,  0.0384749...])
```

CIE 1994 chromatic adaptation, requires extra *kwargs*:

```
>>> XYZ = np.array([0.2800, 0.2126, 0.0527])
>>> XYZ_w = np.array([1.09867452, 1.00000000, 0.35591556])
>>> XYZ_wr = np.array([0.95045593, 1.00000000, 1.08905775])
>>> Y_o = 0.20
>>> E_o = 1000
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method='CIE 1994', Y_o=Y_o, E_o1=E_o, E_o2=E_o)
...
array([ 0.2403379...,  0.2115621...,  0.1764301...])
```

CMCCAT2000 chromatic adaptation, requires extra *kwargs*:

```
>>> XYZ = np.array([0.2248, 0.2274, 0.0854])
>>> XYZ_w = np.array([1.1115, 1.0000, 0.3520])
>>> XYZ_wr = np.array([0.9481, 1.0000, 1.0730])
>>> L_A = 200
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method='CMCCAT2000', L_A1=L_A, L_A2=L_A)
...
array([ 0.1952698...,  0.2306834...,  0.2497175...])
```

Fairchild (1990) chromatic adaptation, requires extra *kwargs*:

```
>>> XYZ = np.array([0.1953, 0.2307, 0.2497])
>>> Y_n = 200
>>> chromatic_adaptation(
...     XYZ, XYZ_w, XYZ_wr, method='Fairchild 1990', Y_n=Y_n)
...
array([ 0.2332526...,  0.2332455...,  0.7611593...])
```

colour.CHROMATIC_ADAPTATION_METHODS

colour.**CHROMATIC_ADAPTATION_METHODS = CaseInsensitiveMapping({'CIE 1994': ..., 'CMCCAT2000': ..., 'Fairc**

Supported chromatic adaptation methods.

References

[CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

CHROMATIC_ADAPTATION_METHODS [CaseInsensitiveMapping] **{'CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries'}**

colour.CMCCAT2000_VIEWING_CONDITIONS

colour.**CMCCAT2000_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...}**
Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[LLRH02], [WRC12a]

CMCCAT2000_VIEWING_CONDITIONS [CaseInsensitiveMapping] ('Average', 'Dim', 'Dark')

Dataset

colour

CHROMATIC_ADAPTATION_TRANSFORMS	Supported chromatic adaptation transforms.

colour.CHROMATIC_ADAPTATION_TRANSFORMS

colour.**CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({'XYZ Scaling': ..., 'Von Kries': ..., '**
Supported chromatic adaptation transforms.

References

[BS10], [BS08], [Fai], [LPLMv07], [Lin09a], [WRC12b], [WRC12a], [Wik07a]

CHROMATIC_ADAPTATION_TRANSFORMS [CaseInsensitiveMapping] **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild, 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**

Fairchild (1990)

colour.adaptation

chromatic_adaptation_Fairchild1990(XYZ_1, ...)	Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *Fairchild (1990)* chromatic adaptation model.

colour.adaptation.chromatic_adaptation_Fairchild1990

colour.adaptation.**chromatic_adaptation_Fairchild1990**(*XYZ_1*, *XYZ_n*, *XYZ_r*, *Y_n*, *discount_illuminant=False*)

Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *Fairchild (1990)* chromatic adaptation model.

Parameters

- **XYZ_1** (array_like) – *CIE XYZ_1* tristimulus values of test sample / stimulus.
- **XYZ_n** (array_like) – Test viewing condition *CIE XYZ_n* tristimulus values of whitepoint.
- **XYZ_r** (array_like) – Reference viewing condition *CIE XYZ_r* tristimulus values of whitepoint.
- **Y_n** (numeric or array_like) – Luminance Y_n of test adapting stimulus in cd/m^2.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns Adapted *CIE XYZ_2* tristimulus values of stimulus.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ_1	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]
XYZ_r	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_2	[0, 100]	[0, 1]

References

[Fai91], [Fai13c]

Examples

```
>>> XYZ_1 = np.array([19.53, 23.07, 24.97])
>>> XYZ_n = np.array([111.15, 100.00, 35.20])
>>> XYZ_r = np.array([94.81, 100.00, 107.30])
>>> Y_n = 200
>>> chromatic_adaptation_Fairchild1990(XYZ_1, XYZ_n, XYZ_r, Y_n)
...
array([ 23.3252634...,  23.3245581...,  76.1159375...])
```

CIE 1994

colour.adaptation

chromatic_adaptation_CIE1994(XYZ_1, xy_o1, ...)	Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *CIE 1994* chromatic adaptation model.

colour.adaptation.chromatic_adaptation_CIE1994

colour.adaptation.**chromatic_adaptation_CIE1994**(*XYZ_1*, *xy_o1*, *xy_o2*, *Y_o*, *E_o1*, *E_o2*, *n=1*)

Adapts given stimulus *CIE XYZ_1* tristimulus values from test viewing conditions to reference viewing conditions using *CIE 1994* chromatic adaptation model.

Parameters

- **XYZ_1** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **xy_o1** (array_like) – Chromaticity coordinates x_{o1} and y_{o1} of test illuminant and background.
- **xy_o2** (array_like) – Chromaticity coordinates x_{o2} and y_{o2} of reference illuminant and background.
- **Y_o** (numeric) – Luminance factor Y_o of achromatic background as percentage normalised to domain [18, 100] in **'Reference'** domain-range scale.
- **E_o1** (numeric) – Test illuminance E_{o1} in cd/m^2.
- **E_o2** (numeric) – Reference illuminance E_{o2} in cd/m^2.
- **n** (numeric, optional) – Noise component in fundamental primary system.

Returns Adapted *CIE XYZ_2* tristimulus values of test stimulus.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ_1	[0, 100]	[0, 1]
Y_o	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_2	[0, 100]	[0, 1]

References

[CIET13294]

Examples

```
>>> XYZ_1 = np.array([28.00, 21.26, 5.27])
>>> xy_o1 = np.array([0.4476, 0.4074])
>>> xy_o2 = np.array([0.3127, 0.3290])
>>> Y_o = 20
>>> E_o1 = 1000
>>> E_o2 = 1000
>>> chromatic_adaptation_CIE1994(XYZ_1, xy_o1, xy_o2, Y_o, E_o1, E_o2)
...
array([ 24.0337952...,  21.1562121...,  17.6430119...])
```

CMCCAT2000

colour.adaptation

chromatic_adaptation_CMCCAT2000(XYZ, XYZ_w, ...)	Adapts given stimulus *CIE XYZ* tristimulus values using given viewing conditions.
CMCCAT2000_VIEWING_CONDITIONS	Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

colour.adaptation.chromatic_adaptation_CMCCAT2000

colour.adaptation.**chromatic_adaptation_CMCCAT2000**(*XYZ*, *XYZ_w*, *XYZ_wr*, *L_A1*, *L_A2*, *surround=CMCCAT2000_InductionFactors(F=1)*, *direction='Forward'*)

Adapts given stimulus *CIE XYZ* tristimulus values using given viewing conditions.

This definition is a convenient wrapper around colour.adaptation.chromatic_adaptation_forward_CMCCAT2000() and colour.adaptation.chromatic_adaptation_inverse_CMCCAT2000().

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Source viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (array_like) – Target viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (numeric or array_like) – Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (numeric or array_like) – Luminance of reference adapting field L_{A2} in cd/m^2.
- **surround** (CMCCAT2000_InductionFactors, optional) – Surround viewing conditions induction factors.
- **direction** (unicode, optional) – **{'Forward', 'Inverse'}**, Chromatic adaptation direction.

Returns Adapted stimulus *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_wr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[LLRH02], [WRC12a]

Examples

```
>>> XYZ = np.array([22.48, 22.74, 8.54])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_CMCCAT2000(
...     XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, direction='Forward')
...
array([ 19.5269832...,  23.0683396...,  24.9717522...])
```

Using the *CMCCAT2000* inverse model:

```
>>> XYZ = np.array([19.52698326, 23.06833960, 24.97175229])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_CMCCAT2000(
...     XYZ, XYZ_w, XYZ_wr, L_A1, L_A2, direction='Inverse')
...
array([ 22.48,  22.74,   8.54])
```

colour.adaptation.CMCCAT2000_VIEWING_CONDITIONS

colour.adaptation.**CMCCAT2000_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'D**

Reference *CMCCAT2000* chromatic adaptation model viewing conditions.

References

[LLRH02], [WRC12a]

CMCCAT2000_VIEWING_CONDITIONS [CaseInsensitiveMapping] ('Average', 'Dim', 'Dark')

Ancillary Objects

colour.adaptation

chromatic_adaptation_forward_CMCCAT2000(XYZ, ...)	Adapts given stimulus *CIE XYZ* tristimulus values from test viewing conditions to reference viewing conditions using *CMCCAT2000* forward chromatic adaptation model.
chromatic_adaptation_inverse_CMCCAT2000(...)	Adapts given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* inverse chromatic adaptation model.
CMCCAT2000_InductionFactors	*CMCCAT2000* chromatic adaptation model induction factors.

colour.adaptation.chromatic_adaptation_forward_CMCCAT2000

colour.adaptation.**chromatic_adaptation_forward_CMCCAT2000**(*XYZ*, *XYZ_w*, *XYZ_wr*, *L_A1*, *L_A2*, *surround=CMCCAT2000_InductionFactors(F=1)*)

Adapts given stimulus *CIE XYZ* tristimulus values from test viewing conditions to reference viewing conditions using *CMCCAT2000* forward chromatic adaptation model.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (numeric or array_like) – Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (numeric or array_like) – Luminance of reference adapting field L_{A2} in cd/m^2.
- **surround** (CMCCAT2000_InductionFactors, optional) – Surround viewing conditions induction factors.

Returns *CIE XYZ_c* tristimulus values of the stimulus corresponding colour.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_wr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_c	[0, 100]	[0, 1]

References

[LLRH02], [WRC12a]

Examples

```
>>> XYZ = np.array([22.48, 22.74, 8.54])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_forward_CMCCAT2000(XYZ, XYZ_w, XYZ_wr, L_A1, L_A2)
...
array([ 19.5269832...,  23.0683396...,  24.9717522...])
```

colour.adaptation.chromatic_adaptation_inverse_CMCCAT2000

colour.adaptation.**chromatic_adaptation_inverse_CMCCAT2000**(*XYZ_c*, *XYZ_w*, *XYZ_wr*, *L_A1*, *L_A2*, *surround=CMCCAT2000_InductionFactors(F=1)*)

Adapts given stimulus corresponding colour *CIE XYZ* tristimulus values from reference viewing conditions to test viewing conditions using *CMCCAT2000* inverse chromatic adaptation model.

Parameters

- **XYZ_c** (array_like) – *CIE XYZ* tristimulus values of the stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of the whitepoint.
- **L_A1** (numeric or array_like) – Luminance of test adapting field L_{A1} in cd/m^2.
- **L_A2** (numeric or array_like) – Luminance of reference adapting field L_{A2} in cd/m^2.
- **surround** (CMCCAT2000_InductionFactors, optional) – Surround viewing conditions induction factors.

Returns *CIE XYZ_c* tristimulus values of the adapted stimulus.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ_c	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_wr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[LLRH02], [WRC12a]

Examples

```
>>> XYZ_c = np.array([19.53, 23.07, 24.97])
>>> XYZ_w = np.array([111.15, 100.00, 35.20])
>>> XYZ_wr = np.array([94.81, 100.00, 107.30])
>>> L_A1 = 200
>>> L_A2 = 200
>>> chromatic_adaptation_inverse_CMCCAT2000(XYZ_c, XYZ_w, XYZ_wr, L_A1,
...                                         L_A2)
...
array([ 22.4839876...,  22.7419485...,   8.5393392...])
```

colour.adaptation.CMCCAT2000_InductionFactors

class colour.adaptation.**CMCCAT2000_InductionFactors**

CMCCAT2000 chromatic adaptation model induction factors.

Parameters **F** (numeric or array_like) – F surround condition.

References

[LLRH02], [WRC12a]

Create new instance of CMCCAT2000_InductionFactors(F,)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

F	Alias for field number 0

Von Kries

colour.adaptation

chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)	Adapts given stimulus from test viewing conditions to reference viewing conditions.
CHROMATIC_ADAPTATION_TRANSFORMS	Supported chromatic adaptation transforms.

colour.adaptation.chromatic_adaptation_VonKries

colour.adaptation.**chromatic_adaptation_VonKries**(*XYZ*, *XYZ_w*, *XYZ_wr*, *transform='CAT02'*)

Adapts given stimulus from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of stimulus to adapt.
- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of whitepoint.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of whitepoint.
- **transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns *CIE XYZ_c* tristimulus values of the stimulus corresponding colour.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
XYZ_n	[0, 1]	[0, 1]
XYZ_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ_c	[0, 1]	[0, 1]

References

[Fai13b]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr)
array([ 0.2163881...,  0.1257    ,  0.0384749...])
```

Using Bradford method:

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> transform = 'Bradford'
>>> chromatic_adaptation_VonKries(XYZ, XYZ_w, XYZ_wr, transform)
...
array([ 0.2166600...,  0.1260477...,  0.0385506...])
```

colour.adaptation.CHROMATIC_ADAPTATION_TRANSFORMS

colour.adaptation.**CHROMATIC_ADAPTATION_TRANSFORMS = CaseInsensitiveMapping({'XYZ Scaling': ..., 'Von Krie**

Supported chromatic adaptation transforms.

References

[BS10], [BS08], [Fai], [LPLMv07], [Lin09a], [WRC12b], [WRC12a], [Wik07a]

CHROMATIC_ADAPTATION_TRANSFORMS [CaseInsensitiveMapping] **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**

Dataset

colour.adaptation

BRADFORD_CAT
BS_CAT
BS_PC_CAT
CAT02_BRILL_CAT
CAT02_CAT
CMCCAT2000_CAT
CMCCAT97_CAT
FAIRCHILD_CAT
SHARP_CAT
VON_KRIES_CAT
XYZ_SCALING_CAT

colour.adaptation.BRADFORD_CAT

colour.adaptation.**BRADFORD_CAT = array([[0.8951, 0.2664, -0.1614], [-0.7502, 1.7135, 0.0367], [0.0389,**

colour.adaptation.BS_CAT

colour.adaptation.**BS_CAT = array([[0.8752, 0.2787, -0.1539], [-0.8904, 1.8709, 0.0195], [-0.0061, 0.0162**

colour.adaptation.BS_PC_CAT

colour.adaptation.**BS_PC_CAT = array([[0.6489, 0.3915, -0.0404], [-0.3775, 1.3055, 0.072], [-0.0271, 0.0**

colour.adaptation.CAT02_BRILL_CAT

colour.adaptation.**CAT02_BRILL_CAT = array([[0.7328, 0.4296, -0.1624], [-0.7036, 1.6975, 0.0061], [0. ,**

colour.adaptation.CAT02_CAT

colour.adaptation.**CAT02_CAT = array([[0.7328, 0.4296, -0.1624], [-0.7036, 1.6975, 0.0061], [0.003 , 0.0**

colour.adaptation.CMCCAT2000_CAT

colour.adaptation.**CMCCAT2000_CAT = array([[7.98200000e-01, 3.38900000e-01, -1.37100000e-01], [-5.918000**

colour.adaptation.CMCCAT97_CAT

colour.adaptation.**CMCCAT97_CAT = array([[0.8951, -0.7502, 0.0389], [0.2664, 1.7135, 0.0685], [-0.1614,**

colour.adaptation.FAIRCHILD_CAT

colour.adaptation.**FAIRCHILD_CAT = array([[0.8562, 0.3372, -0.1934], [-0.836 , 1.8327, 0.0033], [0.0357,**

colour.adaptation.SHARP_CAT

colour.adaptation.**SHARP_CAT = array([[1.2694, -0.0988, -0.1706], [-0.8364, 1.8006, 0.0357], [0.0297, -0**

colour.adaptation.VON_KRIES_CAT

colour.adaptation.**VON_KRIES_CAT = array([[0.40024, 0.7076 , -0.08081], [-0.2263 , 1.16532, 0.0457], [0**

colour.adaptation.XYZ_SCALING_CAT

colour.adaptation.**XYZ_SCALING_CAT = array([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])**

Ancillary Objects

colour.adaptation

chromatic_adaptation_matrix_VonKries(XYZ_w, ...)	Computes the *chromatic adaptation* matrix from test viewing conditions to reference viewing conditions.

colour.adaptation.chromatic_adaptation_matrix_VonKries

colour.adaptation.**chromatic_adaptation_matrix_VonKries**(*XYZ_w*, *XYZ_wr*, *transform='CAT02'*)

Computes the *chromatic adaptation* matrix from test viewing conditions to reference viewing conditions.

Parameters

- **XYZ_w** (array_like) – Test viewing condition *CIE XYZ* tristimulus values of whitepoint.
- **XYZ_wr** (array_like) – Reference viewing condition *CIE XYZ* tristimulus values of whitepoint.
- **transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT',**

'Bianco', 'Bianco PC'}, Chromatic adaptation transform.

Returns Chromatic adaptation matrix M_{cat}.

Return type ndarray

Raises `KeyError` – If chromatic adaptation method is not defined.

Notes

Domain	Scale - Reference	Scale - 1
XYZ_w	[0, 1]	[0, 1]
XYZ_wr	[0, 1]	[0, 1]

References

[Fai13b]

Examples

```
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr)
...
array([[ 1.0425738...,  0.0308910..., -0.0528125...],
       [ 0.0221934...,  1.0018566..., -0.0210737...],
       [-0.0011648..., -0.0034205...,  0.7617890...]])
```

Using Bradford method:

```
>>> XYZ_w = np.array([0.95045593, 1.00000000, 1.08905775])
>>> XYZ_wr = np.array([0.96429568, 1.00000000, 0.82510460])
>>> method = 'Bradford'
>>> chromatic_adaptation_matrix_VonKries(XYZ_w, XYZ_wr, method)
...
array([[ 1.0479297...,  0.0229468..., -0.0501922...],
       [ 0.0296278...,  0.9904344..., -0.0170738...],
       [-0.0092430...,  0.0150551...,  0.7518742...]])
```

Algebra

- *Extrapolation*
- *Interpolation*
- *Coordinates*
- *Geometry*
- *Matrix*
- *Random*
- *Regression*
- *Common*

Extrapolation

colour

Extrapolator([interpolator, method, left, ...])	Extrapolates the 1-D function of given interpolator.

colour.Extrapolator

class colour.**Extrapolator**(*interpolator=None*, *method='Linear'*, *left=None*, *right=None*, *dtype=<class 'numpy.float64'>*)

Bases: object

Extrapolates the 1-D function of given interpolator.

The colour.Extrapolator class acts as a wrapper around a given *Colour* or *scipy* interpolator class instance with compatible signature. Two extrapolation methods are available:

- *Linear*: Linearly extrapolates given points using the slope defined by the interpolator boundaries (xi[0], xi[1]) if x < xi[0] and (xi[-1], xi[-2]) if x > xi[-1].
- *Constant*: Extrapolates given points by assigning the interpolator boundaries values xi[0] if x < xi[0] and xi[-1] if x > xi[-1].

Specifying the *left* and *right* arguments takes precedence on the chosen extrapolation method and will assign the respective *left* and *right* values to the given points.

Parameters

- **interpolator** (object) – Interpolator object.
- **method** (unicode, optional) – **{'Linear', 'Constant'}**, Extrapolation method.
- **left** (numeric, optional) – Value to return for x < xi[0].
- **right** (numeric, optional) – Value to return for x > xi[-1].
- **dtype** (type) – Data type used for internal conversions.

__class__()

Notes

- The interpolator must define *x* and *y* attributes.

References

[Sas], [WRC12d]

Examples

Extrapolating a single numeric variable:

```
>>> from colour.algebra import LinearInterpolator
>>> x = np.array([3, 4, 5])
>>> y = np.array([1, 2, 3])
>>> interpolator = LinearInterpolator(x, y)
>>> extrapolator = Extrapolator(interpolator)
>>> extrapolator(1)
-1.0
```

Extrapolating an *array_like* variable:

```
>>> extrapolator(np.array([6, 7 , 8]))
array([ 4.,  5.,  6.])
```

Using the *Constant* extrapolation method:

```
>>> x = np.array([3, 4, 5])
>>> y = np.array([1, 2, 3])
>>> interpolator = LinearInterpolator(x, y)
>>> extrapolator = Extrapolator(interpolator, method='Constant')
>>> extrapolator(np.array([0.1, 0.2, 8, 9]))
array([ 1.,  1.,  3.,  3.])
```

Using defined *left* boundary and *Constant* extrapolation method:

```
>>> x = np.array([3, 4, 5])
>>> y = np.array([1, 2, 3])
>>> interpolator = LinearInterpolator(x, y)
>>> extrapolator = Extrapolator(interpolator, method='Constant', left=0)
>>> extrapolator(np.array([0.1, 0.2, 8, 9]))
array([ 0.,  0.,  3.,  3.])
```

property interpolator
Getter and setter property for the *Colour* or *scipy* interpolator class instance.

Parameters **value** (callable) – Value to set the *Colour* or *scipy* interpolator class instance with.

Returns *Colour* or *scipy* interpolator class instance.

Return type callable

property left
Getter and setter property for left value to return for x < xi[0].

Parameters **value** (numeric) – Left value to return for x < xi[0].

Returns Left value to return for x < xi[0].

Return type numeric

property method
Getter and setter property for the extrapolation method.

Parameters **value** (unicode) – Value to set the extrapolation method. with.

Returns Extrapolation method.

Return type unicode

property right
Getter and setter property for right value to return for x > xi[-1].

Parameters **value** (numeric) – Right value to return for x > xi[-1].

Returns Right value to return for x > xi[-1].

Return type numeric

Interpolation

colour

KernelInterpolator(x, y[, window, kernel, ...])	Kernel based interpolation of a 1-D function.
NearestNeighbourInterpolator(*args, **kwargs)	A nearest-neighbour interpolator.
LinearInterpolator(x, y[, dtype])	Linearly interpolates a 1-D function.
NullInterpolator(x, y[, absolute_tolerance, ...])	Performs 1-D function null interpolation, i.e.
PchipInterpolator(x, y, *args, **kwargs)	Interpolates a 1-D function using Piecewise Cubic Hermite Interpolating Polynomial interpolation.
SpragueInterpolator(x, y[, dtype])	Constructs a fifth-order polynomial that passes through y dependent variable.

colour.KernelInterpolator

class colour.**KernelInterpolator**(*x*, *y*, *window=3*, *kernel=<function kernel_lanczos>*, *kernel_args=None*, *padding_args=None*, *dtype=<class 'numpy.float64'>*)

Bases: object

Kernel based interpolation of a 1-D function.

The reconstruction of a continuous signal can be described as a linear convolution operation. Interpolation can be expressed as a convolution of the given discrete function $g(x)$ with some continuous interpolation kernel $k(w)$:

$\hat{g}(w_0) = [k * g](w_0) = \sum_{x=-\infty}^{\infty} k(w_0 - x) \cdot g(x)$

Parameters

- **x** (array_like) – Independent x variable values corresponding with y variable.
- **y** (array_like) – Dependent and already known y variable values to interpolate.
- **window** (int, optional) – Width of the window in samples on each side.
- **kernel** (callable, optional) – Kernel to use for interpolation.
- **kernel_args** (dict, optional) – Arguments to use when calling the kernel.
- **padding_args** (dict, optional) – Arguments to use when padding y variable values with the np.pad() definition.
- **dtype** (type) – Data type used for internal conversions.

x

y

window

kernel

kernel_args

padding_args

__call__()

References

[BB09], [Wik05c]

Examples

Interpolating a single numeric variable:

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = KernelInterpolator(x, y)
>>> f(0.5)
6.9411400...
```

Interpolating an *array_like* variable:

```
>>> f([0.25, 0.75])
array([ 6.1806208...,  8.0823848...])
```

Using a different *lanczos* kernel:

```
>>> f = KernelInterpolator(x, y, kernel=kernel_sinc)
>>> f([0.25, 0.75])
array([ 6.5147317...,  8.3965466...])
```

Using a different window size:

```
>>> f = KernelInterpolator(
...     x,
...     y,
...     window=16,
...     kernel=kernel_lanczos,
...     kernel_args={'a': 16})
>>> f([0.25, 0.75])
array([ 5.3961792...,  5.6521093...])
```

property kernel
Getter and setter property for the kernel callable.

Parameters **value** (callable) – Value to set the kernel callable.

Returns Kernel callable.

Return type callable

property kernel_args
Getter and setter property for the kernel call time arguments.

Parameters **value** (dict) – Value to call the interpolation kernel with.

Returns Kernel call time arguments.

Return type dict

property padding_args
Getter and setter property for the kernel call time arguments.

Parameters **value** (dict) – Value to call the interpolation kernel with.

Returns Kernel call time arguments.

Return type dict

property window
Getter and setter property for the window.

Parameters **value** (int) – Value to set the window with.

Returns Window.

Return type int

property x

Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y

Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

colour.NearestNeighbourInterpolator

class colour.**NearestNeighbourInterpolator**(**args*, ***kwargs*)

Bases: colour.algebra.interpolation.KernelInterpolator

A nearest-neighbour interpolator.

Other Parameters

- **x** (*array_like*) – Independent x variable values corresponding with y variable.
- **y** (*array_like*) – Dependent and already known y variable values to interpolate.
- **window** (*int, optional*) – Width of the window in samples on each side.
- **padding_args** (*dict, optional*) – Arguments to use when padding y variable values with the np.pad() definition.
- **dtype** (*type*) – Data type used for internal conversions.

colour.LinearInterpolator

class colour.**LinearInterpolator**(*x*, *y*, *dtype=<class 'numpy.float64'>*)

Bases: object

Linearly interpolates a 1-D function.

Parameters

- **x** (array_like) – Independent x variable values corresponding with y variable.
- **y** (array_like) – Dependent and already known y variable values to interpolate.
- **dtype** (type) – Data type used for internal conversions.

x

y

__call__()

Notes

- This class is a wrapper around *numpy.interp* definition.

Examples

Interpolating a single numeric variable:

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = LinearInterpolator(x, y)
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> f(0.5)
7.64...
```

Interpolating an *array_like* variable:

```
>>> f([0.25, 0.75])
array([ 6.7825,  8.5075])
```

property x
Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y
Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

colour.NullInterpolator

class colour.**NullInterpolator**(*x*, *y*, *absolute_tolerance=1e-06*, *relative_tolerance=1e-06*, *default=nan*, *dtype=<class 'numpy.float64'>*)
Bases: object

Performs 1-D function null interpolation, i.e. a call within given tolerances will return existing y variable values and default if outside tolerances.

Parameters

- **x** (ndarray) – Independent x variable values corresponding with y variable.
- **y** (ndarray) – Dependent and already known y variable values to interpolate.
- **absolute_tolerance** (numeric, optional) – Absolute tolerance.
- **relative_tolerance** (numeric, optional) – Relative tolerance.
- **default** (numeric, optional) – Default value for interpolation outside tolerances.
- **dtype** (type) – Data type used for internal conversions.

x

y

relative_tolerance

absolute_tolerance

default

__call__()

Examples

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = NullInterpolator(x, y)
>>> f(0.5)
nan
>>> f(1.0)
9.3699999...
>>> f = NullInterpolator(x, y, absolute_tolerance=0.01)
>>> f(1.01)
9.3699999...
```

property absolute_tolerance
Getter and setter property for the absolute tolerance.

Parameters **value** (numeric) – Value to set the absolute tolerance with.

Returns Absolute tolerance.

Return type numeric

property default
Getter and setter property for the default value for call outside tolerances.

Parameters **value** (numeric) – Value to set the default value with.

Returns Default value.

Return type numeric

property relative_tolerance
Getter and setter property for the relative tolerance.

Parameters **value** (numeric) – Value to set the relative tolerance with.

Returns Relative tolerance.

Return type numeric

property x
Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y
Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

colour.PchipInterpolator

class colour.**PchipInterpolator**(*x*, *y*, **args*, ***kwargs*)
Bases: scipy.interpolate._cubic.PchipInterpolator

Interpolates a 1-D function using Piecewise Cubic Hermite Interpolating Polynomial interpolation.

y

Notes

- This class is a wrapper around *scipy.interpolate.PchipInterpolator* class.

property y
Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

colour.SpragueInterpolator

class colour.**SpragueInterpolator**(*x*, *y*, *dtype=<class 'numpy.float64'>*)
Bases: object

Constructs a fifth-order polynomial that passes through y dependent variable.

Sprague (1880) method is recommended by the *CIE* for interpolating functions having a uniformly spaced independent variable.

Parameters

- **x** (array_like) – Independent x variable values corresponding with y variable.
- **y** (array_like) – Dependent and already known y variable values to interpolate.
- **dtype** (type) – Data type used for internal conversions.

x

y

__call__()

Notes

- The minimum number k of data points required along the interpolation axis is $k = 6$.

References

[CIET13805b], [WRC12e]

Examples

Interpolating a single numeric variable:

```
>>> y = np.array([5.9200, 9.3700, 10.8135, 4.5100,
...               69.5900, 27.8007, 86.0500])
>>> x = np.arange(len(y))
>>> f = SpragueInterpolator(x, y)
>>> f(0.5)
7.2185025...
```

Interpolating an *array_like* variable:

```
>>> f([0.25, 0.75])
array([ 6.7295161...,  7.8140625...])
```

SPRAGUE_C_COEFFICIENTS = array([[884, -1960, 3033, -2648, 1080, -180], [508, -540, 488, -367, 144,

Defines the coefficients used to generate extra points for boundaries interpolation.

SPRAGUE_C_COEFFICIENTS : array_like, (4, 6)

References

[CIET13805d]

property x

Getter and setter property for the independent x variable.

Parameters **value** (array_like) – Value to set the independent x variable with.

Returns Independent x variable.

Return type array_like

property y

Getter and setter property for the dependent and already known y variable.

Parameters **value** (array_like) – Value to set the dependent and already known y variable with.

Returns Dependent and already known y variable.

Return type array_like

lagrange_coefficients(r[, n])	Computes the *Lagrange Coefficients* at given point r for degree n.
TABLE_INTERPOLATION_METHODS	Supported table interpolation methods.
table_interpolation(V_xyz, table[, method])	Performs interpolation of given V_{xyz} values using given interpolation table.

colour.lagrange_coefficients

colour.**lagrange_coefficients**(*r*, *n=4*)

Computes the *Lagrange Coefficients* at given point r for degree n.

Parameters

- **r** (numeric) – Point to get the *Lagrange Coefficients* at.
- **n** (int, optional) – Degree of the *Lagrange Coefficients* being calculated.

Returns

Return type ndarray

References

[Fai85], [Wik03b]

Examples

```
>>> lagrange_coefficients(0.1)
array([ 0.8265,  0.2755, -0.1305,  0.0285])
```

colour.TABLE_INTERPOLATION_METHODS

colour.**TABLE_INTERPOLATION_METHODS = CaseInsensitiveMapping({'Trilinear': ..., 'Tetrahedral': ...})**

Supported table interpolation methods.

References

[Boub], [Kir06]

TABLE_INTERPOLATION_METHODS [CaseInsensitiveMapping] **{'Trilinear', 'Tetrahedral'}**

colour.table_interpolation

colour.**table_interpolation**(*V_xyz*, *table*, *method='Trilinear'*)

Performs interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_xyz** (array_like) – V_{xyz} values to interpolate.
- **table** (array_like) – 4-Dimensional (NxNxNx3) interpolation table.
- **method** (unicode, optional) – **{'Trilinear', 'Tetrahedral'}**, Interpolation method.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[Boub], [Kir06]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),'..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'Colour_Correct.cube')
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation(V_xyz, table)
array([[ 1.0120664...,  0.7539146...,  1.0228540...],
       [ 0.5075794...,  0.6479459...,  0.1066404...],
       [ 1.0976519...,  0.1785998...,  0.2299897...]])
>>> table_interpolation(V_xyz, table, method='Tetrahedral')
...
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Interpolation Kernels

colour

kernel_nearest_neighbour(x)	Returns the *nearest-neighbour* kernel evaluated at given samples.
kernel_linear(x)	Returns the *linear* kernel evaluated at given samples.
kernel_sinc(x[, a])	Returns the *sinc* kernel evaluated at given samples.
kernel_lanczos(x[, a])	Returns the *lanczos* kernel evaluated at given samples.
kernel_cardinal_spline(x[, a, b])	Returns the *cardinal spline* kernel evaluated at given samples.

colour.kernel_nearest_neighbour

colour.**kernel_nearest_neighbour**(*x*)

Returns the *nearest-neighbour* kernel evaluated at given samples.

Parameters **x** (array_like) – Samples at which to evaluate the *nearest-neighbour* kernel.

Returns The *nearest-neighbour* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_nearest_neighbour(np.linspace(0, 1, 10))
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0])
```

colour.kernel_linear

colour.**kernel_linear**(*x*)

Returns the *linear* kernel evaluated at given samples.

Parameters **x** (array_like) – Samples at which to evaluate the *linear* kernel.

Returns The *linear* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_linear(np.linspace(0, 1, 10))
array([ 1.        ,  0.8888888...,  0.7777777...,  0.6666666...,  0.5555555...,
        0.4444444...,  0.3333333...,  0.2222222...,  0.1111111...,  0.        ])
```

colour.kernel_sinc

colour.**kernel_sinc**(*x*, *a=3*)

Returns the *sinc* kernel evaluated at given samples.

Parameters

- **x** (array_like) – Samples at which to evaluate the *sinc* kernel.
- **a** (int, optional) – Size of the *sinc* kernel.

Returns The *sinc* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_sinc(np.linspace(0, 1, 10))
array([  1.0000000...e+00,   9.7981553...e-01,   9.2072542...e-01,
         8.2699334...e-01,   7.0531659...e-01,   5.6425327...e-01,
         4.1349667...e-01,   2.6306440...e-01,   1.2247694...e-01,
         3.8981718...e-17])
```

colour.kernel_lanczos

colour.**kernel_lanczos**(*x*, *a=3*)

Returns the *lanczos* kernel evaluated at given samples.

Parameters

- **x** (array_like) – Samples at which to evaluate the *lanczos* kernel.
- **a** (int, optional) – Size of the *lanczos* kernel.

Returns The *lanczos* kernel evaluated at given samples.

Return type ndarray

References

[Wik05c]

Examples

```
>>> kernel_lanczos(np.linspace(0, 1, 10))
array([  1.0000000...e+00,   9.7760615...e-01,   9.1243770...e-01,
         8.1030092...e-01,   6.8012706...e-01,   5.3295773...e-01,
         3.8071690...e-01,   2.3492839...e-01,   1.0554054...e-01,
         3.2237621...e-17])
```

colour.kernel_cardinal_spline

colour.**kernel_cardinal_spline**(*x*, *a=0.5*, *b=0.0*)

Returns the *cardinal spline* kernel evaluated at given samples.

Notable *cardinal spline* a and b parameterizations:

- *Catmull-Rom*: $(a = 0.5, b = 0)$
- *Cubic B-Spline*: $(a = 0, b = 1)$
- *Mitchell-Netravalli*: $(a = \frac{1}{3}, b = \frac{1}{3})$

Parameters

- **x** (array_like) – Samples at which to evaluate the *cardinal spline* kernel.
- **a** (int, optional) – a control parameter.
- **b** (int, optional) – b control parameter.

Returns The *cardinal spline* kernel evaluated at given samples.

Return type ndarray

References

[BB09]

Examples

```
>>> kernel_cardinal_spline(np.linspace(0, 1, 10))
array([ 1.        ,  0.9711934...,  0.8930041...,  0.7777777...,  0.6378600...,
        0.4855967...,  0.3333333...,  0.1934156...,  0.0781893...,  0.        ])
```

Ancillary Objects

colour.algebra

table_interpolation_trilinear(V_xyz, table)	Performs trilinear interpolation of given V_{xyz} values using given interpolation table.
table_interpolation_tetrahedral(V_xyz, table)	Performs tetrahedral interpolation of given V_{xyz} values using given interpolation table.

colour.algebra.table_interpolation_trilinear

colour.algebra.**table_interpolation_trilinear**(*V_xyz*, *table*)

Performs trilinear interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_xyz** (array_like) – V_{xyz} values to interpolate.
- **table** (array_like) – 4-Dimensional (NxNxNx3) interpolation table.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[Boub]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),'..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'Colour_Correct.cube')
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_trilinear(V_xyz, table)
array([[ 1.0120664...,  0.7539146...,  1.0228540...],
       [ 0.5075794...,  0.6479459...,  0.1066404...],
       [ 1.0976519...,  0.1785998...,  0.2299897...]])
```

colour.algebra.table_interpolation_tetrahedral

colour.algebra.**table_interpolation_tetrahedral**(*V_xyz*, *table*)

Performs tetrahedral interpolation of given V_{xyz} values using given interpolation table.

Parameters

- **V_xyz** (array_like) – V_{xyz} values to interpolate.
- **table** (array_like) – 4-Dimensional (NxNxNx3) interpolation table.

Returns Interpolated V_{xyz} values.

Return type ndarray

References

[Kir06]

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(
...     os.path.dirname(__file__),'..', 'io', 'luts', 'tests', 'resources',
...     'iridas_cube', 'Colour_Correct.cube')
>>> LUT = colour.read_LUT(path)
>>> table = LUT.table
>>> prng = np.random.RandomState(4)
>>> V_xyz = colour.algebra.random_triplet_generator(3, random_state=prng)
>>> print(V_xyz)
[[ 0.9670298...  0.7148159...  0.9762744...]
 [ 0.5472322...  0.6977288...  0.0062302...]
 [ 0.9726843...  0.2160895...  0.2529823...]]
>>> table_interpolation_tetrahedral(V_xyz, table)
array([[ 1.0196197...,  0.7674062...,  1.0311751...],
       [ 0.5105603...,  0.6466722...,  0.1077296...],
       [ 1.1178206...,  0.1762039...,  0.2209534...]])
```

Coordinates

colour.algebra

cartesian_to_spherical(a)	Transforms given Cartesian coordinates array xyz to Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth).
spherical_to_cartesian(a)	Transforms given Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to Cartesian coordinates array xyz.
cartesian_to_polar(a)	Transforms given Cartesian coordinates array xy to Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).
polar_to_cartesian(a)	Transforms given Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to Cartesian coordinates array xy.

Continued on next page

Table 17 – continued from previous page

cartesian_to_cylindrical(a)	Transforms given Cartesian coordinates array xyz to Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height).
cylindrical_to_cartesian(a)	Transforms given Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height) to Cartesian coordinates array xyz.

colour.algebra.cartesian_to_spherical

colour.algebra.**cartesian_to_spherical**(*a*)

Transforms given Cartesian coordinates array xyz to Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth).

Parameters **a** (array_like) – Cartesian coordinates array xyz to transform.

Returns Spherical coordinates array $\rho\theta\phi$.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_spherical(a)
array([ 6.7823299...,  1.0857465...,  0.3217505...])
```

colour.algebra.spherical_to_cartesian

colour.algebra.**spherical_to_cartesian**(*a*)

Transforms given Spherical coordinates array $\rho\theta\phi$ (radial distance, inclination or elevation and azimuth) to Cartesian coordinates array xyz.

Parameters **a** (array_like) – Spherical coordinates array $\rho\theta\phi$ to transform.

Returns Cartesian coordinates array xyz.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([6.78232998, 1.08574654, 0.32175055])
>>> spherical_to_cartesian(a)
array([ 3.        ,  0.9999999...,  6.        ])
```

colour.algebra.cartesian_to_polar

colour.algebra.**cartesian_to_polar**(a)

Transforms given Cartesian coordinates array xy to Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate).

Parameters **a** (array_like) – Cartesian coordinates array xy to transform.

Returns Polar coordinates array $\rho\phi$.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3, 1])
>>> cartesian_to_polar(a)
array([ 3.1622776...,  0.3217505...])
```

colour.algebra.polar_to_cartesian

colour.algebra.**polar_to_cartesian**(a)

Transforms given Polar coordinates array $\rho\phi$ (radial coordinate, angular coordinate) to Cartesian coordinates array xy.

Parameters **a** (array_like) – Polar coordinates array $\rho\phi$ to transform.

Returns Cartesian coordinates array xy.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3.16227766, 0.32175055])
>>> polar_to_cartesian(a)
array([ 3.        ,  0.9999999...])
```

colour.algebra.cartesian_to_cylindrical

colour.algebra.**cartesian_to_cylindrical**(a)

Transforms given Cartesian coordinates array xyz to Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height).

Parameters **a** (array_like) – Cartesian coordinates array xyz to transform.

Returns Cylindrical coordinates array $\rho\phi z$.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3, 1, 6])
>>> cartesian_to_cylindrical(a)
array([ 3.1622776...,  0.3217505...,  6.        ])
```

colour.algebra.cylindrical_to_cartesian

colour.algebra.**cylindrical_to_cartesian**(a)

Transforms given Cylindrical coordinates array $\rho\phi z$ (azimuth, radial distance and height) to Cartesian coordinates array xyz.

Parameters **a** (array_like) – Cylindrical coordinates array $\rho\phi z$ to transform.

Returns Cartesian coordinates array xyz.

Return type ndarray

References

[Wik06a], [Wik05b]

Examples

```
>>> a = np.array([3.16227766, 0.32175055, 6.00000000])
>>> cylindrical_to_cartesian(a)
array([ 3.        ,  0.9999999...,  6.        ])
```

Geometry

colour.algebra

normalise_vector(a)	Normalises given vector a.
euclidean_distance(a, b)	Returns the euclidean distance between point arrays a and b.
extend_line_segment(a, b[, distance])	Extends the line segment defined by point arrays a and b by given distance and return the new end point.

Continued on next page

Table 18 – continued from previous page

intersect_line_segments(l_1, l_2)	Computes l_1 line segments intersections with l_2 line segments.
ellipse_coefficients_general_form(coefficients)	Returns the general form ellipse coefficients from given canonical form ellipse coefficients.
ellipse_coefficients_canonical_form(coefficients)	Returns the canonical form ellipse coefficients from given general form ellipse coefficients.
point_at_angle_on_ellipse(phi, coefficients)	Returns the coordinates of the point at angle ϕ in degrees on the ellipse with given canonical form coefficients.
ELLIPSE_FITTING_METHODS	Supported ellipse fitting methods.
ellipse_fitting(a[, method])	Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using given method.

colour.algebra.normalise_vector

colour.algebra.**normalise_vector**(a)

Normalises given vector a.

Parameters **a** (array_like) – Vector a to normalise.

Returns Normalised vector a.

Return type ndarray

Examples

```
>>> a = np.array([0.20654008, 0.12197225, 0.05136952])
>>> normalise_vector(a)
array([ 0.8419703...,  0.4972256...,  0.2094102...])
```

colour.algebra.euclidean_distance

colour.algebra.**euclidean_distance**(a, b)

Returns the euclidean distance between point arrays a and b.

Parameters

- **a** (array_like) – Point array a.
- **b** (array_like) – Point array b.

Returns Euclidean distance.

Return type numeric or ndarray

Examples

```
>>> a = np.array([100.00000000, 21.57210357, 272.22819350])
>>> b = np.array([100.00000000, 426.67945353, 72.39590835])
>>> euclidean_distance(a, b)
451.7133019...
```

colour.algebra.extend_line_segment

colour.algebra.**extend_line_segment**(*a*, *b*, *distance=1*)

Extends the line segment defined by point arrays a and b by given distance and return the new end point.

Parameters

- **a** (array_like) – Point array a.
- **b** (array_like) – Point array b.
- **distance** (numeric, optional) – Distance to extend the line segment.

Returns New end point.

Return type ndarray

References

[Sae]

Notes

- Input line segment points coordinates are 2d coordinates.

Examples

```
>>> a = np.array([0.95694934, 0.13720932])
>>> b = np.array([0.28382835, 0.60608318])
>>> extend_line_segment(a, b)
array([-0.5367248...,  1.1776534...])
```

colour.algebra.intersect_line_segments

colour.algebra.**intersect_line_segments**(*l_1*, *l_2*)

Computes l_1 line segments intersections with l_2 line segments.

Parameters

- **l_1** (array_like) – l_1 line segments array, each row is a line segment such as (x_1, y_1, x_2, y_2) where (x_1, y_1) and (x_2, y_2) are respectively the start and end points of l_1 line segments.
- **l_2** (array_like) – l_2 line segments array, each row is a line segment such as (x_3, y_3, x_4, y_4) where (x_3, y_3) and (x_4, y_4) are respectively the start and end points of l_2 line segments.

Returns Line segments intersections specification.

Return type *LineSegmentsIntersections_Specification*

References

[Boua], [Erda]

Notes

- Input line segments points coordinates are 2d coordinates.

Examples

```
>>> l_1 = np.array(
...     [[[0.15416284, 0.7400497],
...       [0.26331502, 0.53373939]],
...      [[0.01457496, 0.91874701],
...       [0.90071485, 0.03342143]]]
... )
>>> l_2 = np.array(
...     [[[0.95694934, 0.13720932],
...       [0.28382835, 0.60608318]],
...       [[0.94422514, 0.85273554],
...       [0.00225923, 0.52122603]],
...       [[0.55203763, 0.48537741],
...       [0.76813415, 0.16071675]]]
... )
>>> s = intersect_line_segments(l_1, l_2)
>>> s.xy
array([[[        nan,         nan],
        [ 0.2279184...,  0.6006430...],
        [        nan,         nan]],

       [[ 0.4281451...,  0.5055568...],
        [ 0.3056055...,  0.6279838...],
        [ 0.7578749...,  0.1761301...]]])
>>> s.intersect
array([[False,  True, False],
       [ True,  True,  True]], dtype=bool)
>>> s.parallel
array([[False, False, False],
       [False, False, False]], dtype=bool)
>>> s.coincident
array([[False, False, False],
       [False, False, False]], dtype=bool)
```

colour.algebra.ellipse_coefficients_general_form

colour.algebra.**ellipse_coefficients_general_form**(*coefficients*)

Returns the general form ellipse coefficients from given canonical form ellipse coefficients.

The canonical form ellipse coefficients are as follows: the center coordinates x_c and y_c, semi-major axis length a_a, semi-minor axis length a_b and rotation angle θ in degrees of its semi-major axis a_a.

Parameters **coefficients** (array_like) – Canonical form ellipse coefficients.

Returns General form ellipse coefficients.

Return type ndarray

References

[Wik]

Examples

```
>>> coefficients = np.array([0.5, 0.5, 2, 1, 45])
>>> ellipse_coefficients_general_form(coefficients)
array([ 2.5, -3. ,  2.5, -1. , -1. , -3.5])
```

colour.algebra.ellipse_coefficients_canonical_form

colour.algebra.**ellipse_coefficients_canonical_form**(*coefficients*)

Returns the canonical form ellipse coefficients from given general form ellipse coefficients.

The general form ellipse coefficients are the coefficients of the implicit second-order polynomial/quadratic curve expressed as follows:

$F(x, y) = \text{ax\^{}2 + bxy + cy\^{}2 + dx + ey + f = 0`}$

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters **coefficients** (array_like) – General form ellipse coefficients.

Returns Canonical form ellipse coefficients.

Return type ndarray

References

[Wik]

Examples

```
>>> coefficients = np.array([ 2.5, -3.0,  2.5, -1.0, -1.0, -3.5])
>>> ellipse_coefficients_canonical_form(coefficients)
array([  0.5,   0.5,   2. ,   1. ,  45. ])
```

colour.algebra.point_at_angle_on_ellipse

colour.algebra.**point_at_angle_on_ellipse**(*phi*, *coefficients*)

Returns the coordinates of the point at angle ϕ in degrees on the ellipse with given canonical form coefficients.

Parameters

- **phi** (array_like) – Point at angle ϕ in degrees to retrieve the coordinates of.
- **coefficients** (array_like) – General form ellipse coefficients as follows: the center coordinates x_c and y_c, semi-major axis length a_a, semi-minor axis length a_b and rotation angle θ in degrees of its semi-major axis a_a.

Returns Coordinates of the point at angle ϕ

Return type ndarray

Examples

```
>>> coefficients = np.array([0.5, 0.5, 2, 1, 45])
>>> point_at_angle_on_ellipse(45, coefficients)
array([ 1.,  2.])
```

colour.algebra.ELLIPSE_FITTING_METHODS

colour.algebra.**ELLIPSE_FITTING_METHODS = CaseInsensitiveMapping({'Halir 1998': ...})**
Supported ellipse fitting methods.

References

[HF98]

ELLIPSE_FITTING_METHODS [CaseInsensitiveMapping] **{'Halir 1998'}**

colour.algebra.ellipse_fitting

colour.algebra.**ellipse_fitting**(*a*, *method='Halir 1998'*)
Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using given method.

The implicit second-order polynomial is expressed as follows:

```
:math:`F\left(x, y\right)` = ax^2 + bxy + cy^2 + dx + ey + f = 0`
```

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters

- **a** (array_like) – Point array a to be fitted.
- **method** (unicode, optional) – **{'Halir 1998'}**, Computation method.

Returns Coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a.

Return type ndarray

References

[HF98]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting(a)
array([ 0.2425356...,  0.        ,  0.9701425...,  0.        ,  0.        ,
       -0.9701425...])
>>> ellipse_coefficients_canonical_form(ellipse_fitting(a))
array([-0., -0.,  2.,  1.,  0.])
```

Ancillary Objects

colour.algebra

LineSegmentsIntersections_Specification	Defines the specification for intersection of line segments l_1 and l_2 returned by colour.algebra.intersect_line_segments() definition.
ellipse_fitting_Halir1998(a)	Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using *Halir and Flusser (1998)* method.

colour.algebra.LineSegmentsIntersections_Specification

class colour.algebra.**LineSegmentsIntersections_Specification**

Defines the specification for intersection of line segments l_1 and l_2 returned by colour.algebra.intersect_line_segments() definition.

Parameters

- **xy** (array_like) – Array of l_1 and l_2 line segments intersections coordinates. Non existing segments intersections coordinates are set with *np.nan*.
- **intersect** (array_like) – Array of *bool* indicating if line segments l_1 and l_2 intersect.
- **parallel** (array_like) – Array of *bool* indicating if line segments l_1 and l_2 are parallel.
- **coincident** (array_like) – Array of *bool* indicating if line segments l_1 and l_2 are coincident.

Create new instance of LineSegmentsIntersections_Specification(xy, intersect, parallel, coincident)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

coincident	Alias for field number 3
intersect	Alias for field number 1
parallel	Alias for field number 2
xy	Alias for field number 0

colour.algebra.ellipse_fitting_Halir1998

colour.algebra.**ellipse_fitting_Halir1998**(*a*)

Returns the coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a using *Halir and Flusser (1998)* method.

The implicit second-order polynomial is expressed as follows:

```
:math:`F\left(x, y\right)` = ax^2 + bxy + cy^2 + dx + ey + f = 0`
```

with an ellipse-specific constraint such as $b^2 - 4ac < 0$ and where a, b, c, d, e, f are coefficients of the ellipse and $F(x, y)$ are coordinates of points lying on it.

Parameters **a** (array_like) – Point array a to be fitted.

Returns Coefficients of the implicit second-order polynomial/quadratic curve that fits given point array a.

Return type ndarray

References

[HF98]

Examples

```
>>> a = np.array([[2, 0], [0, 1], [-2, 0], [0, -1]])
>>> ellipse_fitting_Halir1998(a)
array([ 0.2425356...,  0.        ,  0.9701425...,  0.        ,  0.        ,
       -0.9701425...])
>>> ellipse_coefficients_canonical_form(ellipse_fitting_Halir1998(a))
array([-0., -0.,  2.,  1.,  0.])
```

Matrix

colour.algebra

is_identity(a[, n])	Returns if a array is an identity matrix.

colour.algebra.is_identity

colour.algebra.**is_identity**(*a*, *n=3*)

Returns if a array is an identity matrix.

Parameters

- **a** (array_like, (N)) – Variable a to test.
- **n** (int, optional) – Matrix dimension.

Returns Is identity matrix.

Return type bool

Examples

```
>>> is_identity(np.array([1, 0, 0, 0, 1, 0, 0, 0, 1]).reshape(3, 3))
True
>>> is_identity(np.array([1, 2, 0, 0, 1, 0, 0, 0, 1]).reshape(3, 3))
False
```

Random

colour.algebra

random_triplet_generator(size[, limits, ...])	Returns a generator yielding random triplets.

colour.algebra.random_triplet_generator

colour.algebra.**random_triplet_generator**(*size*, *limits=array([[0, 1], [0, 1], [0, 1]])*, *random_state=RandomState(MT19937) at 0x7F03837CFCA8*)

Returns a generator yielding random triplets.

Parameters

- **size** (int) – Generator size.
- **limits** (array_like, (3, 2)) – Random values limits on each triplet axis.
- **random_state** (RandomState) – Mersenne Twister pseudo-random number generator.

Returns Random triplets generator.

Return type generator

Notes

- The test is assuming that np.random.RandomState() definition will return the same sequence no matter which *OS* or *Python* version is used. There is however no formal promise about the *prng* sequence reproducibility of either *Python* or *Numpy* implementations, see [Lau12].

Examples

```
>>> from pprint import pprint
>>> prng = np.random.RandomState(4)
>>> random_triplet_generator(10, random_state=prng)
...
array([[ 0.9670298...,  0.7793829...,  0.4361466...],
       [ 0.5472322...,  0.1976850...,  0.9489773...],
       [ 0.9726843...,  0.8629932...,  0.7863059...],
       [ 0.7148159...,  0.9834006...,  0.8662893...],
       [ 0.6977288...,  0.1638422...,  0.1731654...],
       [ 0.2160895...,  0.5973339...,  0.0749485...],
       [ 0.9762744...,  0.0089861...,  0.6007427...],
       [ 0.0062302...,  0.3865712...,  0.1679721...],
       [ 0.2529823...,  0.0441600...,  0.7333801...],
       [ 0.4347915...,  0.9566529...,  0.4084438...]])
```

Regression

colour.algebra

least_square_mapping_MoorePenrose(y, x)	Computes the *least-squares* mapping from dependent variable y to independent variable x using *Moore-Penrose* inverse.

colour.algebra.least_square_mapping_MoorePenrose

colour.algebra.**least_square_mapping_MoorePenrose**(y, x)

Computes the *least-squares* mapping from dependent variable y to independent variable x using *Moore-Penrose* inverse.

Parameters

- **y** (array_like) – Dependent and already known y variable.
- **x** (array_like, optional) – Independent x variable(s) values corresponding with y variable.

Returns *Least-squares* mapping.

Return type ndarray

References

[FMH15]

Examples

```
>>> prng = np.random.RandomState(2)
>>> y = prng.random_sample((24, 3))
>>> x = y + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> least_square_mapping_MoorePenrose(y, x)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

Common

colour.algebra

is_spow_enabled()	Returns whether *Colour* safe / symmetrical power function is enabled.
set_spow_enable(enable)	Sets *Colour* safe / symmetrical power function enabled state.
spow_enable(enable)	A context manager and decorator temporarily setting *Colour* safe / symmetrical power function enabled state.
spow(a, p)	Raises given array a to the power p as follows: $sign(a) * |a|^p$.

colour.algebra.is_spow_enabled

colour.algebra.**is_spow_enabled**()
Returns whether *Colour* safe / symmetrical power function is enabled.

Returns Whether *Colour* safe / symmetrical power function is enabled.

Return type bool

Examples

```
>>> with spow_enable(False):
...     is_spow_enabled()
False
>>> with spow_enable(True):
...     is_spow_enabled()
True
```

colour.algebra.set_spow_enable

colour.algebra.**set_spow_enable**(*enable*)
Sets *Colour* safe / symmetrical power function enabled state.

Parameters **enable** (bool) – Whether to enable *Colour* safe / symmetrical power function.

Examples

```
>>> with spow_enable(is_spow_enabled()):
...     print(is_spow_enabled())
...     set_spow_enable(False)
...     print(is_spow_enabled())
True
False
```

colour.algebra.spow_enable

class colour.algebra.**spow_enable**(*enable*)
A context manager and decorator temporarily setting *Colour* safe / symmetrical power function enabled state.

Parameters **enable** (bool) – Whether to enable or disable *Colour* safe / symmetrical power function.

__init__(*enable*)
Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(enable)	Initialize self.

colour.algebra.spow

colour.algebra.**spow**(a, p)

Raises given array a to the power p as follows: $sign(a) * |a|^p$.

This definition avoids NaNs generation when array a is negative and the power p is fractional. This behaviour can be enabled or disabled with the `colour.algebra.set_spow_enable()` definition or with the `spow_enable()` context manager.

Parameters

- **a** (numeric or array_like) – Array a.
- **p** (numeric or array_like) – Power p.

Returns Array a safely raised to the power p.

Return type numeric or ndarray

Examples

```
>>> np.power(-2, 0.15)
nan
>>> spow(-2, 0.15)
-1.1095694...
>>> spow(0, 0)
0.0
```

Colour Appearance Models

- *ATD (1995)*
- *CIECAM02*
- *CAM16*
- *Hunt*
- $LLAB(l : c)$
- *Nayatani (1995)*
- *RLAB*

ATD (1995)

colour

XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2[, sigma])	Computes the *ATD (1995)* colour vision model correlates.
ATD95_Specification	Defines the *ATD (1995)* colour vision model specification.

colour.XYZ_to_ATD95

colour.**XYZ_to_ATD95**(*XYZ*, *XYZ_0*, *Y_0*, *k_1*, *k_2*, *sigma=300*)

Computes the *ATD (1995)* colour vision model correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **Y_0** (numeric or array_like) – Absolute adapting field luminance in cd/m^2.
- **k_1** (numeric or array_like) – Application specific weight k_1.
- **k_2** (numeric or array_like) – Application specific weight k_2.
- **sigma** (numeric or array_like, optional) – Constant σ varied to predict different types of data.

Returns *ATD (1995)* colour vision model specification.

Return type *ATD95_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
ATD95_Specification.h	[0, 360]	[0, 1]

- For unrelated colors, there is only self-adaptation and k_1 is set to 1.0 while k_2 is set to 0.0. For related colors such as typical colorimetric applications, k_1 is set to 0.0 and k_2 is set to a value between 15 and 50 *(Guth, 1995)*.

References

[Fai13a], [Gut95]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_0 = np.array([95.05, 100.00, 108.88])
>>> Y_0 = 318.31
>>> k_1 = 0.0
>>> k_2 = 50.0
>>> XYZ_to_ATD95(XYZ, XYZ_0, Y_0, k_1, k_2)
ATD95_Specification(h=1.9089869..., C=1.2064060..., Q=0.1814003..., A_1=0.1787931... T_1=0.
↪0286942..., D_1=0.0107584..., A_2=0.0192182..., T_2=0.0205377..., D_2=0.0107584...)
```

colour.ATD95_Specification

class colour.**ATD95_Specification**

Defines the *ATD (1995)* colour vision model specification.

This specification has field names consistent with the remaining colour appearance models in `colour.appearance` but diverge from *Fairchild (2013)* reference.

Parameters

- **h** (`numeric or array_like`) – *Hue* angle H in degrees.
- **C** (`numeric or array_like`) – Correlate of *saturation* C. *Guth (1995)* incorrectly uses the terms saturation and chroma interchangeably. However, C is here a measure of saturation rather than chroma since it is measured relative to the achromatic response for the stimulus rather than that of a similarly illuminated white.
- **Q** (`numeric or array_like`) – Correlate of *brightness* Br.
- **A_1** (`numeric or array_like`) – First stage A_1 response.
- **T_1** (`numeric or array_like`) – First stage T_1 response.
- **D_1** (`numeric or array_like`) – First stage D_1 response.
- **A_2** (`numeric or array_like`) – Second stage A_2 response.
- **T_2** (`numeric or array_like`) – Second stage A_2 response.
- **D_2** (`numeric or array_like`) – Second stage D_2 response.

Notes

- This specification is the one used in the current model implementation.

References

[Fai13a], [Gut95]

Create new instance of ATD95_Specification(h, C, Q, A_1, T_1, D_1, A_2, T_2, D_2)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

A_1	Alias for field number 3
A_2	Alias for field number 6
C	Alias for field number 1
D_1	Alias for field number 5
D_2	Alias for field number 8
Q	Alias for field number 2
T_1	Alias for field number 4
T_2	Alias for field number 7
h	Alias for field number 0

CIECAM02

colour

XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b[, ...])	Computes the *CIECAM02* colour appearance model correlates from given *CIE XYZ* tristimulus values.
CIECAM02_to_XYZ(CIECAM02_specification, ...)	Converts from *CIECAM02* specification to *CIE XYZ* tristimulus values.
CIECAM02_Specification	Defines the *CIECAM02* colour appearance model specification.
CIECAM02_VIEWING_CONDITIONS	Reference *CIECAM02* colour appearance model viewing conditions.

colour.XYZ_to_CIECAM02

colour.**XYZ_to_CIECAM02**(*XYZ*, *XYZ_w*, *L_A*, *Y_b*, *surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1)*, *discount_illuminant=False*)

Computes the *CIECAM02* colour appearance model correlates from given *CIE XYZ* tristimulus values.

This is the *forward* implementation.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2.
- **surround** (CIECAM02_InductionFactors, optional) – Surround viewing conditions induction factors.

- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns *CIECAM02* colour appearance model specification.

Return type *CIECAM02_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
CIECAM02_specification.J	[0, 100]	[0, 1]
CIECAM02_specification.C	[0, 100]	[0, 1]
CIECAM02_specification.h	[0, 360]	[0, 1]
CIECAM02_specification.s	[0, 100]	[0, 1]
CIECAM02_specification.Q	[0, 100]	[0, 1]
CIECAM02_specification.M	[0, 100]	[0, 1]
CIECAM02_specification.H	[0, 360]	[0, 1]

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b, surround)
CIECAM02_Specification(J=41.7310911..., C=0.1047077..., h=219.0484326..., s=2.3603053...,␣
↪Q=195.3713259..., M=0.1088421..., H=278.0607358..., HC=None)
```

colour.CIECAM02_to_XYZ

colour.**CIECAM02_to_XYZ**(*CIECAM02_specification*, *XYZ_w*, *L_A*, *Y_b*, *surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1)*, *discount_illuminant=False*)

Converts from *CIECAM02* specification to *CIE XYZ* tristimulus values.

This is the *inverse* implementation.

Parameters

- **CIECAM02_specification** (CIECAM02_Specification) – *CIECAM02* colour appearance model specification. Correlate of *Lightness* J, correlate of *chroma* C or correlate of *colourfulness* M and *hue* angle h in degrees must be specified, e.g. JCh or JMh.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.

- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2.
- **surround** (CIECAM02_InductionFactors, optional) – Surround viewing conditions.
- **discount_illuminant** (bool, optional) – Discount the illuminant.

Returns **XYZ** – *CIE XYZ* tristimulus values.

Return type ndarray

Raises **ValueError** – If neither *C* or *M* correlates have been defined in the CIECAM02_specification argument.

Warning: The output range of that definition is non standard!

Notes

Domain	Scale - Reference	Scale - 1
CIECAM02_specification.J	[0, 100]	[0, 1]
CIECAM02_specification.C	[0, 100]	[0, 1]
CIECAM02_specification.h	[0, 360]	[0, 1]
CIECAM02_specification.s	[0, 100]	[0, 1]
CIECAM02_specification.Q	[0, 100]	[0, 1]
CIECAM02_specification.M	[0, 100]	[0, 1]
CIECAM02_specification.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- CIECAM02_specification can also be passed as a compatible argument to colour.utilities.as_namedtuple() definition.

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Examples

```
>>> specification = CIECAM02_Specification(J=41.731091132513917,
...                                        C=0.104707757171031,
...                                        h=219.048432658311780)
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CIECAM02_to_XYZ(specification, XYZ_w, L_A, Y_b)
array([ 19.01...,  20...  ,  21.78...])
```

colour.CIECAM02_Specification

class colour.**CIECAM02_Specification**

Defines the *CIECAM02* colour appearance model specification.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* J.
- **C** (numeric or array_like) – Correlate of *chroma* C.
- **h** (numeric or array_like) – *Hue* angle h in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s.
- **Q** (numeric or array_like) – Correlate of *brightness* Q.
- **M** (numeric or array_like) – Correlate of *colourfulness* M.
- **H** (numeric or array_like) – *Hue* h quadrature H.
- **HC** (numeric or array_like) – *Hue* h composition H^C.

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Returns a new instance of the colour.CIECAM02_Specification class.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

C	Alias for field number 1
H	Alias for field number 6
HC	Alias for field number 7
J	Alias for field number 0
M	Alias for field number 5
Q	Alias for field number 4
h	Alias for field number 2
s	Alias for field number 3

colour.CIECAM02_VIEWING_CONDITIONS

colour.**CIECAM02_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**
Reference *CIECAM02* colour appearance model viewing conditions.

References

[Fai04], [LL13], [MFH+02], [Wik07b]

CIECAM02_VIEWING_CONDITIONS [CaseInsensitiveMapping] **{'Average', 'Dim', 'Dark'}**

Ancillary Objects

colour.appearance

CIECAM02_InductionFactors	*CIECAM02* colour appearance model induction factors.

colour.appearance.CIECAM02_InductionFactors

class colour.appearance.**CIECAM02_InductionFactors**
CIECAM02 colour appearance model induction factors.

Parameters

- **F** (numeric or array_like) – Maximum degree of adaptation F.
- **c** (numeric or array_like) – Exponential non linearity c.
- **N_c** (numeric or array_like) – Chromatic induction factor N_c.

References

[Fai04], [LL13], [MFH+02], [Wik07b]

Create new instance of CIECAM02_InductionFactors(F, c, N_c)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

F	Alias for field number 0
N_c	Alias for field number 2
c	Alias for field number 1

CAM16

colour

XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b[, ...])	Computes the *CAM16* colour appearance model correlates from given *CIE XYZ* tristimulus values.
CAM16_to_XYZ(CAM16_specification, XYZ_w, ...)	Converts from *CAM16* specification to *CIE XYZ* tristimulus values.
CAM16_Specification	Defines the *CAM16* colour appearance model specification.
CAM16_VIEWING_CONDITIONS	Reference *CAM16* colour appearance model viewing conditions.

colour.XYZ_to_CAM16

colour.**XYZ_to_CAM16**(*XYZ*, *XYZ_w*, *L_A*, *Y_b*, *surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1)*, *discount_illuminant=False*)

Computes the *CAM16* colour appearance model correlates from given *CIE XYZ* tristimulus values.

This is the *forward* implementation.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2.
- **surround** (CAM16_InductionFactors, optional) – Surround viewing conditions induction factors.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns *CAM16* colour appearance model specification.

Return type *CAM16_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
CAM16_specification.J	[0, 100]	[0, 1]
CAM16_specification.C	[0, 100]	[0, 1]
CAM16_specification.h	[0, 360]	[0, 1]
CAM16_specification.s	[0, 100]	[0, 1]
CAM16_specification.Q	[0, 100]	[0, 1]
CAM16_specification.M	[0, 100]	[0, 1]
CAM16_specification.H	[0, 360]	[0, 1]

References

[LLW+17]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CAM16_VIEWING_CONDITIONS['Average']
>>> XYZ_to_CAM16(XYZ, XYZ_w, L_A, Y_b, surround)
CAM16_Specification(J=41.7312079..., C=0.1033557..., h=217.0679597..., s=2.3450150..., Q=195.
↪3717089..., M=0.1074367..., H=275.5949861..., HC=None)
```

colour.CAM16_to_XYZ

colour.**CAM16_to_XYZ**(*CAM16_specification*, *XYZ_w*, *L_A*, *Y_b*, *surround=CIECAM02_InductionFactors(F=1, c=0.69, N_c=1)*, *discount_illuminant=False*)

Converts from *CAM16* specification to *CIE XYZ* tristimulus values.

This is the *inverse* implementation.

Parameters

- **CAM16_specification** (CAM16_Specification) – *CAM16* colour appearance model specification. Correlate of *Lightness* J, correlate of *chroma* C or correlate of *colourfulness* M and *hue* angle h in degrees must be specified, e.g. JCh or JMh.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2, (often taken to be 20% of the luminance of a white object in the scene).
- **Y_b** (numeric or array_like) – Relative luminance of background Y_b in cd/m^2.
- **surround** (CAM16_InductionFactors, optional) – Surround viewing conditions.
- **discount_illuminant** (bool, optional) – Discount the illuminant.

Returns **XYZ** – *CIE XYZ* tristimulus values.

Return type ndarray

Raises **ValueError** – If neither *C* or *M* correlates have been defined in the CAM16_specification argument.

Notes

Domain	Scale - Reference	Scale - 1
CAM16_specification.J	[0, 100]	[0, 1]
CAM16_specification.C	[0, 100]	[0, 1]
CAM16_specification.h	[0, 360]	[0, 1]
CAM16_specification.s	[0, 100]	[0, 1]
CAM16_specification.Q	[0, 100]	[0, 1]
CAM16_specification.M	[0, 100]	[0, 1]
CAM16_specification.H	[0, 360]	[0, 1]
XYZ_w	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- `CAM16_specification` can also be passed as a compatible argument to `colour.utilities.as_namedtuple()` definition.

References

[LLW+17]

Examples

```
>>> specification = CAM16_Specification(J=41.731207905126638,
...                                     C=0.103355738709070,
...                                     h=217.067959767393010)
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> CAM16_to_XYZ(specification, XYZ_w, L_A, Y_b)
array([ 19.01...,  20...  ,  21.78...])
```

colour.CAM16_Specification

class colour.**CAM16_Specification**

Defines the *CAM16* colour appearance model specification.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* J.
- **C** (numeric or array_like) – Correlate of *chroma* C.
- **h** (numeric or array_like) – *Hue* angle h in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s.
- **Q** (numeric or array_like) – Correlate of *brightness* Q.
- **M** (numeric or array_like) – Correlate of *colourfulness* M.
- **H** (numeric or array_like) – *Hue* h quadrature H.
- **HC** (numeric or array_like) – *Hue* h composition H^C.

References

[LLW+17]

Returns a new instance of the `colour.CAM16_Specification` class.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`	Initialize self.
`count`(value)	
`index`(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

`C`	Alias for field number 1
`H`	Alias for field number 6
`HC`	Alias for field number 7
`J`	Alias for field number 0
`M`	Alias for field number 5
`Q`	Alias for field number 4
`h`	Alias for field number 2
`s`	Alias for field number 3

colour.CAM16_VIEWING_CONDITIONS

colour.**CAM16_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**

Reference *CAM16* colour appearance model viewing conditions.

References

[LLW+17]

CAM16_VIEWING_CONDITIONS [CaseInsensitiveMapping] **{'Average', 'Dim', 'Dark'}**

Ancillary Objects

`colour.appearance`

`CAM16_InductionFactors`	*CAM16* colour appearance model induction factors.

colour.appearance.CAM16_InductionFactors

class colour.appearance.**CAM16_InductionFactors**

CAM16 colour appearance model induction factors.

Parameters

- **F** (numeric or array_like) – Maximum degree of adaptation F.
- **c** (numeric or array_like) – Exponential non linearity c.
- **N_c** (numeric or array_like) – Chromatic induction factor N_c.

References

[LLW+17]

Create new instance of CAM16_InductionFactors(F, c, N_c)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

F	Alias for field number 0
N_c	Alias for field number 2
c	Alias for field number 1

Hunt

colour

XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A[, ...])	Computes the *Hunt* colour appearance model correlates.
Hunt_Specification	Defines the *Hunt* colour appearance model specification.
HUNT_VIEWING_CONDITIONS	Reference *Hunt* colour appearance model viewing conditions.

colour.XYZ_to_Hunt

colour.**XYZ_to_Hunt**(*XYZ*, *XYZ_w*, *XYZ_b*, *L_A*, *surround=Hunt_InductionFactors(N_c=1, N_b=75, N_cb=None, N_bb=None)*, *L_AS=None*, *CCT_w=None*, *XYZ_p=None*, *p=None*, *S=None*, *S_w=None*, *helson_judd_effect=False*, *discount_illuminant=True*)
Computes the *Hunt* colour appearance model correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_w** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **XYZ_b** (array_like) – *CIE XYZ* tristimulus values of background.
- **L_A** (numeric or array_like) – Adapting field *luminance* L_A in cd/m^2.
- **surround** (Hunt_InductionFactors, optional) – Surround viewing conditions induction factors.
- **L_AS** (numeric or array_like, optional) – Scotopic luminance L_{AS} of the illuminant, approximated if not specified.

- **CCT_w** (numeric or array_like, optional) – Correlated color temperature T_{cp}: of the illuminant, needed to approximate L_{AS}.
- **XYZ_p** (array_like, optional) – *CIE XYZ* tristimulus values of proximal field, assumed to be equal to background if not specified.
- **p** (numeric or array_like, optional) – Simultaneous contrast / assimilation factor p with value normalised to domain [-1, 0] when simultaneous contrast occurs and normalised to domain [0, 1] when assimilation occurs.
- **S** (numeric or array_like, optional) – Scotopic response S to the stimulus, approximated using tristimulus values Y of the stimulus if not specified.
- **S_w** (numeric or array_like, optional) – Scotopic response S_w for the reference white, approximated using the tristimulus values Y_w of the reference white if not specified.
- **helson_judd_effect** (bool, optional) – Truth value indicating whether the *Helson-Judd* effect should be accounted for.
- **discount_illuminant** (bool, optional) – Truth value indicating if the illuminant should be discounted.

Returns *Hunt* colour appearance model specification.

Return type *Hunt_Specification*

Raises ValueError – If an illegal arguments combination is specified.

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_w	[0, 100]	[0, 1]
XYZ_b	[0, 100]	[0, 1]
XYZ_p	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Hunt_Specification.h	[0, 360]	[0, 1]

References

[Fai13f], [Hun04]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> XYZ_b = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> surround = HUNT_VIEWING_CONDITIONS['Normal Scenes']
>>> CCT_w = 6504.0
>>> XYZ_to_Hunt(XYZ, XYZ_w, XYZ_b, L_A, surround, CCT_w=CCT_w)
...
Hunt_Specification(J=30.0462678..., C=0.1210508..., h=269.2737594..., s=0.0199093..., Q=22.
↪2097654..., M=0.1238964..., H=None, HC=None)
```

colour.Hunt_Specification

class colour.**Hunt_Specification**

Defines the *Hunt* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in `colour.appearance` but diverge from *Fairchild (2013)* reference.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* J.
- **C** (numeric or array_like) – Correlate of *chroma* C_94.
- **h** (numeric or array_like) – *Hue* angle h_S in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s.
- **Q** (numeric or array_like) – Correlate of *brightness* Q.
- **M** (numeric or array_like) – Correlate of *colourfulness* M_94.
- **H** (numeric or array_like) – *Hue* h quadrature H.
- **HC** (numeric or array_like) – *Hue* h composition H_C.

Notes

- This specification is the one used in the current model implementation.

References

[Fai13f], [Hun04]

Create new instance of Hunt_Specification(J, C, h, s, Q, M, H, HC)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

C	Alias for field number 1
H	Alias for field number 6
HC	Alias for field number 7
J	Alias for field number 0
M	Alias for field number 5
Q	Alias for field number 4
h	Alias for field number 2
s	Alias for field number 3

colour.HUNT_VIEWING_CONDITIONS

colour.**HUNT_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Small Areas, Uniform Background & Surrounds': .**

Reference *Hunt* colour appearance model viewing conditions.

References

[Fai13f], [Hun04]

HUNT_VIEWING_CONDITIONS [CaseInsensitiveMapping] **{'Small Areas, Uniform Background & Surrounds', 'Normal Scenes', 'Television & CRT, Dim Surrounds', 'Large Transparencies On Light Boxes', 'Projected Transparencies, Dark Surrounds'}**

Aliases:

- 'small_uniform': 'Small Areas, Uniform Background & Surrounds'
- 'normal': 'Normal Scenes'
- 'tv_dim': 'Television & CRT, Dim Surrounds'
- 'light_boxes': 'Large Transparencies On Light Boxes'
- 'projected_dark': 'Projected Transparencies, Dark Surrounds'

$LLAB(l:c)$

colour

XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L[, surround])	Computes the *:math:`LLAB(l:c)`* colour appearance model correlates.
LLAB_Specification	Defines the *:math:`LLAB(l:c)`* colour appearance model specification.
LLAB_VIEWING_CONDITIONS	Reference $LLAB(l:c)$ colour appearance model viewing conditions.

colour.XYZ_to_LLAB

colour.**XYZ_to_LLAB**(*XYZ*, *XYZ_0*, *Y_b*, *L*, *surround=LLAB_InductionFactors(D=1, F_S=3, F_L=1, F_C=1)*)

Computes the *:math:`LLAB(l:c)`* colour appearance model correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **Y_b** (numeric or array_like) – Luminance factor of the background in cd/m^2.
- **L** (numeric or array_like) – Absolute luminance L of reference white in cd/m^2.
- **surround** (LLAB_InductionFactors, optional) – Surround viewing conditions induction factors.

Returns *:math:`LLAB(l:c)`* colour appearance model specification.

Return type *LLAB_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
LLAB_Specification.h	[0, 360]	[0, 1]

References

[Fai13e], [LLK96], [LM96]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_0 = np.array([95.05, 100.00, 108.88])
>>> Y_b = 20.0
>>> L = 318.31
>>> surround = LLAB_VIEWING_CONDITIONS['ref_average_4_minus']
>>> XYZ_to_LLAB(XYZ, XYZ_0, Y_b, L, surround)
LLAB_Specification(J=37.3668650..., C=0.0089496..., h=270..., s=0.0002395..., M=0.0190185...,␣
↪HC=None, a=..., b=-0.0190185...)
```

colour.LLAB_Specification

class colour.**LLAB_Specification**

Defines the *:math:`LLAB(l:c)`* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in `colour.appearance` but diverge from *Fairchild (2013)* reference.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* L_L.
- **C** (numeric or array_like) – Correlate of *chroma* Ch_L.
- **h** (numeric or array_like) – *Hue* angle h_L in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s_L.
- **M** (numeric or array_like) – Correlate of *colourfulness* C_L.
- **HC** (numeric or array_like) – *Hue* h composition H^C.
- **a** (numeric or array_like) – Opponent signal A_L.
- **b** (numeric or array_like) – Opponent signal B_L.

Notes

- This specification is the one used in the current model implementation.

References

[Fai13e], [LLK96], [LM96]

Create new instance of LLAB_Specification(J, C, h, s, M, HC, a, b)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

C	Alias for field number 1
HC	Alias for field number 5
J	Alias for field number 0
M	Alias for field number 4
a	Alias for field number 6
b	Alias for field number 7
h	Alias for field number 2
s	Alias for field number 3

colour.LLAB_VIEWING_CONDITIONS

colour.**LLAB_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Reference Samples & Images, Average Surround, S**
Reference $LLAB(l:c)$ colour appearance model viewing conditions.

References

[Fai13e], [LLK96], [LM96]

LLAB_VIEWING_CONDITIONS [CaseInsensitiveMapping] **{'Reference Samples & Images, Average Surround, Subtending > 4', 'Reference Samples & Images, Average Surround, Subtending < 4', 'Television & VDU Displays, Dim Surround', 'Cut Sheet Transparency, Dim Surround':, '35mm Projection Transparency, Dark Surround'}**

Aliases:

- 'ref_average_4_plus': 'Reference Samples & Images, Average Surround, Subtending > 4'
- 'ref_average_4_minus': 'Reference Samples & Images, Average Surround, Subtending < 4'
- 'tv_dim': 'Television & VDU Displays, Dim Surround'
- 'sheet_dim': 'Cut Sheet Transparency, Dim Surround'
- 'projected_dark': '35mm Projection Transparency, Dark Surround'

Ancillary Objects

colour.appearance

LLAB_InductionFactors	*:math:`LLAB(l:c)`* colour appearance model induction factors.

colour.appearance.LLAB_InductionFactors

class colour.appearance.**LLAB_InductionFactors**

:math:`LLAB(l:c)` colour appearance model induction factors.

Parameters

- **D** (numeric or array_like) – *Discounting-the-Illuminant* factor D.
- **F_S** (numeric or array_like) – Surround induction factor F_S.
- **F_L** (numeric or array_like) – *Lightness* induction factor F_L.
- **F_C** (numeric or array_like) – *Chroma* induction factor F_C.

References

[Fai13e], [LLK96], [LM96]

Create new instance of LLAB_InductionFactors(D, F_S, F_L, F_C)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

D	Alias for field number 0
F_C	Alias for field number 3
F_L	Alias for field number 2
F_S	Alias for field number 1

Nayatani (1995)

colour

XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or)	Computes the *Nayatani (1995)* colour appearance model correlates.
Nayatani95_Specification	Defines the *Nayatani (1995)* colour appearance model specification.

colour.XYZ_to_Nayatani95

colour.**XYZ_to_Nayatani95**(*XYZ*, *XYZ_n*, *Y_o*, *E_o*, *E_or*, *n=1*)

Computes the *Nayatani (1995)* colour appearance model correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_n** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **Y_o** (numeric or array_like) – Luminance factor Y_o of achromatic background as percentage normalised to domain [0.18, 1.0] in **'Reference'** domain-range scale.
- **E_o** (numeric or array_like) – Illuminance E_o of the viewing field in lux.
- **E_or** (numeric or array_like) – Normalising illuminance E_{or} in lux usually normalised to domain [1000, 3000].
- **n** (numeric or array_like, optional) – Noise term used in the non linear chromatic adaptation model.

Returns *Nayatani (1995)* colour appearance model specification.

Return type *Nayatani95_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Nayatani95_Specification.h	[0, 360]	[0, 1]

References

[Fai13g], [NSY95]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_n = np.array([95.05, 100.00, 108.88])
>>> Y_o = 20.0
>>> E_o = 5000.0
>>> E_or = 1000.0
>>> XYZ_to_Nayatani95(XYZ, XYZ_n, Y_o, E_o, E_or)
Nayatani95_Specification(L_star_P=49.9998829..., C=0.0133550..., h=257.5232268..., s=0.0133550.
↪.., Q=62.6266734..., M=0.0167262..., H=None, HC=None, L_star_N=50.0039154...)
```

colour.Nayatani95_Specification

class colour.**Nayatani95_Specification**

Defines the *Nayatani (1995)* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in `colour.appearance` but diverge from *Fairchild (2013)* reference.

Parameters

- **L_star_P** (numeric or array_like) – Correlate of *achromatic Lightness* $L_p^\star$.
- **C** (numeric or array_like) – Correlate of *chroma* C.
- **h** (numeric or array_like) – *Hue* angle θ in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* S.
- **Q** (numeric or array_like) – Correlate of *brightness* B_r.
- **M** (numeric or array_like) – Correlate of *colourfulness* M.
- **H** (numeric or array_like) – *Hue* h quadrature H.
- **HC** (numeric or array_like) – *Hue* h composition H_C.
- **L_star_N** (numeric or array_like) – Correlate of *normalised achromatic Lightness* $L_n^\star$.

Notes

- This specification is the one used in the current model implementation.

References

[Fai13g], [NSY95]

Create new instance of Nayatani95_Specification(L_star_P, C, h, s, Q, M, H, HC, L_star_N)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

C	Alias for field number 1
H	Alias for field number 6
HC	Alias for field number 7
L_star_N	Alias for field number 8
L_star_P	Alias for field number 0
M	Alias for field number 5
Q	Alias for field number 4
h	Alias for field number 2

Continued on next page

Table 53 – continued from previous page

s	Alias for field number 3

RLAB

colour

XYZ_to_RLAB(XYZ, XYZ_n, Y_n[, sigma, D])	Computes the *RLAB* model color appearance correlates.
RLAB_D_FACTOR	*RLAB* colour appearance model *Discounting-the-Illuminant* factor values.
RLAB_Specification	Defines the *RLAB* colour appearance model specification.
RLAB_VIEWING_CONDITIONS	Reference *RLAB* colour appearance model viewing conditions.

colour.XYZ_to_RLAB

colour.**XYZ_to_RLAB**(*XYZ*, *XYZ_n*, *Y_n*, *sigma=0.4347826086956522*, *D=1*)

Computes the *RLAB* model color appearance correlates.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of test sample / stimulus.
- **XYZ_n** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **Y_n** (numeric or array_like) – Absolute adapting luminance in cd/m^2.
- **sigma** (numeric or array_like, optional) – Relative luminance of the surround, see colour.RLAB_VIEWING_CONDITIONS for reference.
- **D** (numeric or array_like, optional) – *Discounting-the-Illuminant* factor normalised to domain [0, 1].

Returns *RLAB* colour appearance model specification.

Return type *RLAB_Specification*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
RLAB_Specification.h	[0, 360]	[0, 1]

References

[Fai96], [Fai13h]

Examples

```
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_n = np.array([109.85, 100, 35.58])
>>> Y_n = 31.83
>>> sigma = RLAB_VIEWING_CONDITIONS['Average']
>>> D = RLAB_D_FACTOR['Hard Copy Images']
>>> XYZ_to_RLAB(XYZ, XYZ_n, Y_n, sigma, D)
RLAB_Specification(J=49.8347069..., C=54.8700585..., h=286.4860208..., s=1.1010410..., HC=None,
↪ a=15.5711021..., b=-52.6142956...)
```

colour.RLAB_D_FACTOR

colour.**RLAB_D_FACTOR = CaseInsensitiveMapping({'Hard Copy Images': ..., 'Soft Copy Images': ..., 'Projec**

RLAB colour appearance model *Discounting-the-Illuminant* factor values.

References

[Fai96], [Fai13h]

RLAB_D_FACTOR [CaseInsensitiveMapping] **{'Hard Copy Images', 'Soft Copy Images', 'Projected Transparencies, Dark Room'}**

Aliases:

- 'hard_cp_img': 'Hard Copy Images'
- 'soft_cp_img': 'Soft Copy Images'
- 'projected_dark': 'Projected Transparencies, Dark Room'

colour.RLAB_Specification

class colour.**RLAB_Specification**

Defines the *RLAB* colour appearance model specification.

This specification has field names consistent with the remaining colour appearance models in colour.appearance but diverge from *Fairchild (2013)* reference.

Parameters

- **J** (numeric or array_like) – Correlate of *Lightness* L^R.
- **C** (numeric or array_like) – Correlate of *achromatic chroma* C^R.
- **h** (numeric or array_like) – *Hue* angle h^R in degrees.
- **s** (numeric or array_like) – Correlate of *saturation* s^R.
- **HC** (numeric or array_like) – *Hue* h composition H^C.
- **a** (numeric or array_like) – Red-green chromatic response a^R.
- **b** (numeric or array_like) – Yellow-blue chromatic response b^R.

Notes

- This specification is the one used in the current model implementation.

References

[Fai96], [Fai13h]

Create new instance of RLAB_Specification(J, C, h, s, HC, a, b)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

C	Alias for field number 1
HC	Alias for field number 4
J	Alias for field number 0
a	Alias for field number 5
b	Alias for field number 6
h	Alias for field number 2
s	Alias for field number 3

colour.RLAB_VIEWING_CONDITIONS

colour.**RLAB_VIEWING_CONDITIONS = CaseInsensitiveMapping({'Average': ..., 'Dim': ..., 'Dark': ...})**
Reference *RLAB* colour appearance model viewing conditions.

References

[Fai96], [Fai13h]

RLAB_VIEWING_CONDITIONS [CaseInsensitiveMapping] **{'Average', 'Dim', 'Dark'}**

Biochemistry

- *Michaelis–Menten Kinetics*

Michaelis–Menten Kinetics

colour.biochemistry

reaction_rate_MichealisMenten(S, V_max, K_m)	Describes the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S.
substrate_concentration_MichealisMenten(v, ...)	Describes the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v.

colour.biochemistry.reaction_rate_MichealisMenten

colour.biochemistry.**reaction_rate_MichealisMenten**(*S*, *V_max*, *K_m*)

Describes the rate of enzymatic reactions, by relating reaction rate v to concentration of a substrate S.

Parameters

- **S** (array_like) – Concentration of a substrate S.
- **V_max** (array_like) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.
- **K_m** (array_like) – Substrate concentration V_{max} at which the reaction rate is half of V_{max}.

Returns Reaction rate v.

Return type array_like

References

[Wik03e]

Examples

```
>>> reaction_rate_MichealisMenten(0.5, 2.5, 0.8)
0.9615384...
```

colour.biochemistry.substrate_concentration_MichealisMenten

colour.biochemistry.**substrate_concentration_MichealisMenten**(*v*, *V_max*, *K_m*)

Describes the rate of enzymatic reactions, by relating concentration of a substrate S to reaction rate v.

Parameters

- **v** (array_like) – Reaction rate v.
- **V_max** (array_like) – Maximum rate V_{max} achieved by the system, at saturating substrate concentration.
- **K_m** (array_like) – Substrate concentration V_{max} at which the reaction rate is half of V_{max}.

Returns Concentration of a substrate S.

Return type array_like

References

[Wik03e]

Examples

```
>>> substrate_concentration_MichealisMenten(0.961538461538461, 2.5, 0.8)
...
0.4999999...
```

Colour Vision Deficiency

- *Machado, Oliveira and Fernandes (2009)*

Machado, Oliveira and Fernandes (2009)

colour

anomalous_trichromacy_cmfs_Machado2009(cmfs, ...)	Shifts given *LMS* cone fundamentals colour matching functions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy using *Machado et al.(2009)* method..
anomalous_trichromacy_matrix_Machado2009(...)	Computes *Machado et al.(2009) CVD* matrix for given *LMS* cone fundamentals colour matching functions and display primaries tri-spectral distributions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy..
cvd_matrix_Machado2009(deficiency, severity)	Computes *Machado et al.(2009) CVD* matrix for given deficiency and severity using the precomputed matrices dataset..

colour.anomalous_trichromacy_cmfs_Machado2009

colour.**anomalous_trichromacy_cmfs_Machado2009**(*cmfs*, *d_LMS*)

Shifts given *LMS* cone fundamentals colour matching functions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy using *Machado et al. (2009)* method.

Parameters

- **cmfs** (LMS_ConeFundamentals) – *LMS* cone fundamentals colour matching functions.
- **d_LMS** (array_like) – Δ_{LMS} shift amount in nanometers.

Notes

- Input *LMS* cone fundamentals colour matching functions interval is expected to be 1 nanometer, incompatible input will be interpolated at 1 nanometer interval.
- Input Δ_{LMS} shift amount is in domain [0, 20].

Returns Anomalous trichromacy *LMS* cone fundamentals colour matching functions.

Return type *LMS_ConeFundamentals*

Warning: *Machado et al. (2009)* simulation of tritanomaly is based on the shift paradigm as an approximation to the actual phenomenon and restrain the model from trying to model tritanopia. The pre-generated matrices are using a shift value in domain [5, 59] contrary to the domain [0, 20] used for protanomaly and deuteranomaly simulation.

References

[Colb], [Cola], [Colc], [MOF09]

Examples

```
>>> from colour import LMS_CMFS
>>> cmfs = LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> cmfs[450]
array([ 0.0498639,  0.0870524,  0.955393 ])
>>> anomalous_trichromacy_cmfs_Machado2009(cmfs, np.array([15, 0, 0]))[450]
...
array([ 0.0891288...,  0.0870524 ,  0.955393  ])
```

colour.anomalous_trichromacy_matrix_Machado2009

colour.**anomalous_trichromacy_matrix_Machado2009**(*cmfs*, *primaries*, *d_LMS*)

Computes *Machado et al. (2009) CVD* matrix for given *LMS* cone fundamentals colour matching functions and display primaries tri-spectral distributions with given Δ_{LMS} shift amount in nanometers to simulate anomalous trichromacy.

Parameters

- **cmfs** (LMS_ConeFundamentals) – *LMS* cone fundamentals colour matching functions.
- **primaries** (RGB_DisplayPrimaries) – *RGB* display primaries tri-spectral distributions.
- **d_LMS** (array_like) – Δ_{LMS} shift amount in nanometers.

Notes

- Input *LMS* cone fundamentals colour matching functions interval is expected to be 1 nanometer, incompatible input will be interpolated at 1 nanometer interval.
- Input Δ_{LMS} shift amount is in domain [0, 20].

Returns Anomalous trichromacy matrix.

Return type ndarray

References

[Colb], [Cola], [Colc], [MOF09]

Examples

```
>>> from colour import DISPLAYS_RGB_PRIMARIES, LMS_CMFS
>>> cmfs = LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> d_LMS = np.array([15, 0, 0])
>>> primaries = DISPLAYS_RGB_PRIMARIES['Apple Studio Display']
>>> anomalous_trichromacy_matrix_Machado2009(cmfs, primaries, d_LMS)
...
array([[-0.2777465...,  2.6515008..., -1.3737543...],
       [ 0.2718936...,  0.2004786...,  0.5276276...],
       [ 0.0064404...,  0.2592157...,  0.7343437...]])
```

colour.cvd_matrix_Machado2009

colour.**cvd_matrix_Machado2009**(*deficiency*, *severity*)

Computes *Machado et al. (2009) CVD* matrix for given deficiency and severity using the pre-computed matrices dataset.

Parameters

- **deficiency** (unicode) – {'Protanomaly', 'Deuteranomaly', 'Tritanomaly'} Colour blindness / vision deficiency types : - *Protanomaly* : defective long-wavelength cones (L-cones). The complete absence of L-cones is known as *Protanopia* or *red-dichromacy*. - *Deuteranomaly* : defective medium-wavelength cones (M-cones) with peak of sensitivity moved towards the red sensitive cones. The complete absence of M-cones is known as *Deuteranopia*. - *Tritanomaly* : defective short-wavelength cones (S-cones), an alleviated form of blue-yellow color blindness. The complete absence of S-cones is known as *Tritanopia*.
- **severity** (numeric) – Severity of the colour vision deficiency in domain [0, 1].

Returns *CVD* matrix.

Return type ndarray

References

[Colb], [Cola], [Colc], [MOF09]

Examples

```
>>> cvd_matrix_Machado2009('Protanomaly', 0.15)
array([[ 0.7869875...,  0.2694875..., -0.0564735...],
       [ 0.0431695...,  0.933774 ...,  0.023058 ...],
       [-0.004238 ..., -0.0024515...,  1.0066895...]])
```

Dataset

colour

CVD_MATRICES_MACHADO2010	Machado (2010) Simulation matrices Φ_{CVD}.

colour.CVD_MATRICES_MACHADO2010

colour.**CVD_MATRICES_MACHADO2010 = CaseInsensitiveMapping({'Protanomaly': ..., 'Deuteranomaly': ..., 'Tri**

Machado (2010) Simulation matrices Φ_{CVD}.

CVD_MATRICES_MACHADO2010 [CaseInsensitiveMapping] {'Protanomaly', 'Deuteranomaly', 'Tritanomaly'}

Colour Characterisation

- *Colour Fitting*
- *Colour Rendition Charts*
- *Cameras*
- *Displays*

Colour Fitting

colour

POLYNOMIAL_EXPANSION_METHODS	Supported polynomial expansion methods.
polynomial_expansion(a[, method])	Performs polynomial expansion of given a array.
COLOUR_CORRECTION_MATRIX_METHODS	Supported colour correction matrix methods.
colour_correction_matrix(M_T, M_R[, method])	Computes a colour correction matrix from given M_T colour array to M_R colour array.
COLOUR_CORRECTION_METHODS	Supported colour correction methods.
colour_correction(RGB, M_T, M_R[, method])	Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array.

colour.POLYNOMIAL_EXPANSION_METHODS

colour.**POLYNOMIAL_EXPANSION_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015': ...,**
Supported polynomial expansion methods.

References

[CWCR04], [FMH15], [WR04], [Wik03f]

POLYNOMIAL_EXPANSION_METHODS [CaseInsensitiveMapping] **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**

colour.polynomial_expansion

colour.**polynomial_expansion**(*a*, *method='Cheung 2004'*, ***kwargs*)
Performs polynomial expansion of given a array.

Parameters

- **a** (array_like, (3, n)) – a array to expand.
- **method** (unicode, optional) – **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**, Computation method.

Other Parameters

- **degree** (*int*) – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **terms** (*int*) – {colour.characterisation.augmented_matrix_Cheung2004()}, Number of terms of the expanded polynomial, must be one of *[3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22]*.
- **root_polynomial_expansion** (*bool*) – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.

Returns Expanded a array.

Return type ndarray, (3, n)

References

[CWCR04], [FMH15], [WR04], [Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion(RGB)
array([ 0.1722481...,  0.0917066...,  0.0641693...])
>>> polynomial_expansion(RGB, 'Cheung 2004', terms=5)
array([ 0.1722481...,  0.0917066...,  0.0641693...,  0.0010136...,  1...])
```

colour.COLOUR_CORRECTION_MATRIX_METHODS

colour.**COLOUR_CORRECTION_MATRIX_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015':**
Supported colour correction matrix methods.

References

[CWCR04], [FMH15], [WR04], [Wik03f]

POLYNOMIAL_EXPANSION_METHODS [CaseInsensitiveMapping] **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**

colour.colour_correction_matrix

colour.**colour_correction_matrix**(*M_T*, *M_R*, *method='Cheung 2004'*, ***kwargs*)
Computes a colour correction matrix from given M_T colour array to M_R colour array.

The resulting colour correction matrix is computed using multiple linear or polynomial regression using given method. The purpose of that object is for example the matching of two *ColorChecker* colour rendition charts together.

Parameters

- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **method** (unicode, optional) – **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**, Computation method.

Other Parameters

- **degree** (*int*) – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **terms** (*int*) – {colour.characterisation.augmented_matrix_Cheung2004()}, Number of terms of the expanded polynomial, must be one of *[3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22]*.
- **root_polynomial_expansion** (*bool*) – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.

Returns Colour correction matrix.

Return type ndarray, (n, 3)

References

[CWCR04], [FMH15], [WR04], [Wik03f]

Examples

```
>>> M_T = np.array(
...     [[0.17224810, 0.09170660, 0.06416938],
...      [0.49189645, 0.27802050, 0.21923399],
...      [0.10999751, 0.18658946, 0.29938611],
...      [0.11666120, 0.14327905, 0.05713804],
...      [0.18988879, 0.18227649, 0.36056247],
...      [0.12501329, 0.42223442, 0.37027445],
...      [0.64785606, 0.22396782, 0.03365194],
...      [0.06761093, 0.11076896, 0.39779139],
...      [0.49101797, 0.09448929, 0.11623839],
...      [0.11622386, 0.04425753, 0.14469986],
...      [0.36867946, 0.44545230, 0.06028681],
...      [0.61632937, 0.32323906, 0.02437089],
...      [0.03016472, 0.06153243, 0.29014596],
...      [0.11103655, 0.30553067, 0.08149137],
...      [0.41162190, 0.05816656, 0.04845934],
...      [0.73339206, 0.53075188, 0.02475212],
...      [0.47347718, 0.08834792, 0.30310315],
...      [0.00000000, 0.25187016, 0.35062450],
...      [0.76809639, 0.78486240, 0.77808297],
...      [0.53822392, 0.54307997, 0.54710883],
...      [0.35458526, 0.35318419, 0.35524431],
...      [0.17976704, 0.18000531, 0.17991488],
...      [0.09351417, 0.09510603, 0.09675027],
...      [0.03405071, 0.03295077, 0.03702047]]
... )
>>> M_R = np.array(
...     [[0.15579559, 0.09715755, 0.07514556],
...      [0.39113140, 0.25943419, 0.21266708],
...      [0.12824821, 0.18463570, 0.31508023],
...      [0.12028974, 0.13455659, 0.07408400],
...      [0.19368988, 0.21158946, 0.37955964],
...      [0.19957425, 0.36085439, 0.40678123],
...      [0.48896605, 0.20691688, 0.05816533],
...      [0.09775522, 0.16710693, 0.47147724],
...      [0.39358649, 0.12233400, 0.10526425],
...      [0.10780332, 0.07258529, 0.16151473],
...      [0.27502671, 0.34705454, 0.09728099],
...      [0.43980441, 0.26880559, 0.05430533],
...      [0.05887212, 0.11126272, 0.38552469],
...      [0.12705825, 0.25787860, 0.13566464],
...      [0.35612929, 0.07933258, 0.05118732],
...      [0.48131976, 0.42082843, 0.07120612],
...      [0.34665585, 0.15170714, 0.24969804],
...      [0.08261116, 0.24588716, 0.48707733],
...      [0.66054904, 0.65941137, 0.66376412],
...      [0.48051509, 0.47870296, 0.48230082],
...      [0.33045354, 0.32904184, 0.33228886],
...      [0.18001305, 0.17978567, 0.18004416],
...      [0.10283975, 0.10424680, 0.10384975],
...      [0.04742204, 0.04772203, 0.04914226]]
... )
>>> colour_correction_matrix(M_T, M_R)
array([[ 0.6982266...,  0.0307162...,  0.1621042...],
       [ 0.0689349...,  0.6757961...,  0.1643038...],
       [-0.0631495...,  0.0921247...,  0.9713415...]])
```

colour.COLOUR_CORRECTION_METHODS

colour.**COLOUR_CORRECTION_METHODS = CaseInsensitiveMapping({'Cheung 2004': ..., 'Finlayson 2015': ..., 'V**
Supported colour correction methods.

References

[CWCR04], [FMH15], [WR04], [Wik03f]

COLOUR_CORRECTION_METHODS [CaseInsensitiveMapping] **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**

colour.colour_correction

colour.**colour_correction**(*RGB*, *M_T*, *M_R*, *method='Cheung 2004'*, ***kwargs*)
Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array.

Parameters

- **RGB** (array_like, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **method** (unicode, optional) – **{'Cheung 2004', 'Finlayson 2015', 'Vandermonde'}**, Computation method.

Other Parameters

- **degree** (*int*) – {colour.characterisation.polynomial_expansion_Finlayson2015(), colour.characterisation.polynomial_expansion_Vandermonde()}, Expanded polynomial degree, must be one of *[1, 2, 3, 4]* for colour.characterisation.polynomial_expansion_Finlayson2015() definition.
- **terms** (*int*) – {colour.characterisation.augmented_matrix_Cheung2004()}, Number of terms of the expanded polynomial, must be one of *[3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22]*.
- **root_polynomial_expansion** (*bool*) – {colour.characterisation.polynomial_expansion_Finlayson2015()}, Whether to use the root-polynomials set for the expansion.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[CWCR04], [FMH15], [WR04], [Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> M_T = np.array(
...     [[0.17224810, 0.09170660, 0.06416938],
...      [0.49189645, 0.27802050, 0.21923399],
...      [0.10999751, 0.18658946, 0.29938611],
...      [0.11666120, 0.14327905, 0.05713804],
...      [0.18988879, 0.18227649, 0.36056247],
...      [0.12501329, 0.42223442, 0.37027445],
...      [0.64785606, 0.22396782, 0.03365194],
...      [0.06761093, 0.11076896, 0.39779139],
...      [0.49101797, 0.09448929, 0.11623839],
...      [0.11622386, 0.04425753, 0.14469986],
...      [0.36867946, 0.44545230, 0.06028681],
...      [0.61632937, 0.32323906, 0.02437089],
...      [0.03016472, 0.06153243, 0.29014596],
...      [0.11103655, 0.30553067, 0.08149137],
...      [0.41162190, 0.05816656, 0.04845934],
...      [0.73339206, 0.53075188, 0.02475212],
...      [0.47347718, 0.08834792, 0.30310315],
...      [0.00000000, 0.25187016, 0.35062450],
...      [0.76809639, 0.78486240, 0.77808297],
...      [0.53822392, 0.54307997, 0.54710883],
...      [0.35458526, 0.35318419, 0.35524431],
...      [0.17976704, 0.18000531, 0.17991488],
...      [0.09351417, 0.09510603, 0.09675027],
...      [0.03405071, 0.03295077, 0.03702047]]
... )
>>> M_R = np.array(
...     [[0.15579559, 0.09715755, 0.07514556],
...      [0.39113140, 0.25943419, 0.21266708],
...      [0.12824821, 0.18463570, 0.31508023],
...      [0.12028974, 0.13455659, 0.07408400],
...      [0.19368988, 0.21158946, 0.37955964],
...      [0.19957425, 0.36085439, 0.40678123],
...      [0.48896605, 0.20691688, 0.05816533],
...      [0.09775522, 0.16710693, 0.47147724],
...      [0.39358649, 0.12233400, 0.10526425],
...      [0.10780332, 0.07258529, 0.16151473],
...      [0.27502671, 0.34705454, 0.09728099],
...      [0.43980441, 0.26880559, 0.05430533],
...      [0.05887212, 0.11126272, 0.38552469],
...      [0.12705825, 0.25787860, 0.13566464],
...      [0.35612929, 0.07933258, 0.05118732],
...      [0.48131976, 0.42082843, 0.07120612],
...      [0.34665585, 0.15170714, 0.24969804],
...      [0.08261116, 0.24588716, 0.48707733],
...      [0.66054904, 0.65941137, 0.66376412],
...      [0.48051509, 0.47870296, 0.48230082],
...      [0.33045354, 0.32904184, 0.33228886],
...      [0.18001305, 0.17978567, 0.18004416],
...      [0.10283975, 0.10424680, 0.10384975],
...      [0.04742204, 0.04772203, 0.04914226]]
... )
>>> colour_correction(RGB, M_T, M_R)
array([ 0.1334872...,  0.0843921...,  0.0599014...])
```

Ancillary Objects

`colour.characterisation`

augmented_matrix_Cheung2004(RGB[, terms])	Performs polynomial expansion of given *RGB* colourspace array using *Cheung et al.(2004)* method..
polynomial_expansion_Finlayson2015(RGB[, ...])	Performs polynomial expansion of given *RGB* colourspace array using *Finlayson et al.(2015)* method..
polynomial_expansion_Vandermonde(a[, degree])	Performs polynomial expansion of given a array using *Vandermonde* method.
colour_correction_matrix_Cheung2004(M_T, M_R)	Computes a colour correction from given M_T colour array to M_R colour array using *Cheung et al.(2004)* method..
colour_correction_matrix_Finlayson2015(M_T, M_R)	Computes a colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al.(2015)* method..
colour_correction_matrix_Vandermonde(M_T, M_R)	Computes a colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.
colour_correction_Cheung2004(RGB, M_T, M_R)	Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al.(2004)* method..
colour_correction_Finlayson2015(RGB, M_T, M_R)	Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al.(2015)* method..
colour_correction_Vandermonde(RGB, M_T, M_R)	Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.

colour.characterisation.augmented_matrix_Cheung2004

colour.characterisation.**augmented_matrix_Cheung2004**(*RGB*, *terms=3*)

Performs polynomial expansion of given *RGB* colourspace array using *Cheung et al. (2004)* method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to expand.
- **terms** (int, optional) – Number of terms of the expanded polynomial, must be one of *[3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22]*.

Returns Expanded *RGB* colourspace array.

Return type ndarray

Notes

- This definition combines the augmented matrices given in [CWCR04] and [WR04].

References

[CWCR04], [WR04]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> augmented_matrix_Cheung2004(RGB, terms=5)
array([ 0.1722481...,  0.0917066...,  0.0641693...,  0.0010136...,  1...])
```

colour.characterisation.polynomial_expansion_Finlayson2015

colour.characterisation.**polynomial_expansion_Finlayson2015**(*RGB*, *degree=1*, *root_polynomial_expansion=True*)

Performs polynomial expansion of given *RGB* colourspace array using *Finlayson et al. (2015)* method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to expand.
- **degree** (int, optional) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Expanded *RGB* colourspace array.

Return type ndarray

References

[FMH15]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion_Finlayson2015(RGB, degree=2)
array([ 0.1722481...,  0.0917066...,  0.06416938...,  0.0078981...,  0.0029423...,
        0.0055265...])
```

colour.characterisation.polynomial_expansion_Vandermonde

colour.characterisation.**polynomial_expansion_Vandermonde**(*a*, *degree=1*)

Performs polynomial expansion of given a array using *Vandermonde* method.

Parameters

- **a** (array_like) – a array to expand.
- **degree** (int, optional) – Expanded polynomial degree.

Returns Expanded a array.

Return type ndarray

References

[Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> polynomial_expansion_Vandermonde(RGB)
array([ 0.1722481 ,  0.0917066 ,  0.06416938,  1.        ])
```

colour.characterisation.colour_correction_matrix_Cheung2004

colour.characterisation.**colour_correction_matrix_Cheung2004**(*M_T*, *M_R*, *terms=3*)

Computes a colour correction from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.

Parameters

- **M_T** (array_like, (3, n)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (3, n)) – Reference array the array M_T will be colour fitted against.
- **terms** (int, optional) – Number of terms of the expanded polynomial, must be one of *[3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22]*.

Returns Colour correction matrix.

Return type ndarray, (3, n)

References

[CWCR04], [WR04]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_matrix(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.colour_correction_matrix_Finlayson2015

colour.characterisation.**colour_correction_matrix_Finlayson2015**(*M_T*, *M_R*, *degree=1*, *root_polynomial_expansion=True*)

Computes a colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.

Parameters

- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Colour correction matrix.

Return type ndarray, (n, 3)

References

[FMH15]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_matrix(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.colour_correction_matrix_Vandermonde

colour.characterisation.**colour_correction_matrix_Vandermonde**(*M_T*, *M_R*, *degree=1*)

Computes a colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.

Parameters

- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.

Returns Colour correction matrix.

Return type ndarray, (n, 3)

References

[Wik03f]

Examples

```
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_matrix(M_T, M_R)
array([[ 1.0526376...,  0.1378078..., -0.2276339...],
       [ 0.0739584...,  1.0293994..., -0.1060115...],
       [ 0.0572550..., -0.2052633...,  1.1015194...]])
```

colour.characterisation.colour_correction_Cheung2004

colour.characterisation.**colour_correction_Cheung2004**(*RGB*, *M_T*, *M_R*, *terms=3*)

Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Cheung et al. (2004)* method.

Parameters

- **RGB** (array_like, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **terms** (int, optional) – Number of terms of the expanded polynomial, must be one of *[3, 5, 7, 8, 10, 11, 14, 16, 17, 19, 20, 22]*.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[CWCR04], [WR04]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_Cheung2004(RGB, M_T, M_R)
array([ 0.1793456...,  0.1003392...,  0.0617218...])
```

colour.characterisation.colour_correction_Finlayson2015

colour.characterisation.**colour_correction_Finlayson2015**(*RGB*, *M_T*, *M_R*, *degree=1*, *root_polynomial_expansion=True*)

Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Finlayson et al. (2015)* method.

Parameters

- **RGB** (array_like, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.
- **root_polynomial_expansion** (bool) – Whether to use the root-polynomials set for the expansion.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[FMH15]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_Finlayson2015(RGB, M_T, M_R)
array([ 0.1793456...,  0.1003392...,  0.0617218...])
```

colour.characterisation.colour_correction_Vandermonde

colour.characterisation.**colour_correction_Vandermonde**(*RGB*, *M_T*, *M_R*, *degree=1*)

Performs colour correction of given *RGB* colourspace array using the colour correction matrix from given M_T colour array to M_R colour array using *Vandermonde* method.

Parameters

- **RGB** (array_like, (n, 3)) – *RGB* colourspace array to colour correct.
- **M_T** (array_like, (n, 3)) – Test array M_T to fit onto array M_R.
- **M_R** (array_like, (n, 3)) – Reference array the array M_T will be colour fitted against.
- **degree** (int, optional) – Expanded polynomial degree.

Returns Colour corrected *RGB* colourspace array.

Return type ndarray

References

[Wik03f]

Examples

```
>>> RGB = np.array([0.17224810, 0.09170660, 0.06416938])
>>> prng = np.random.RandomState(2)
>>> M_T = prng.random_sample((24, 3))
>>> M_R = M_T + (prng.random_sample((24, 3)) - 0.5) * 0.5
>>> colour_correction_Vandermonde(RGB, M_T, M_R)
array([ 0.2128689...,  0.1106242...,  0.036213 ...])
```

Colour Rendition Charts

Dataset

colour

COLOURCHECKERS	Aggregated *Colour Checkers* chromaticity coordinates.
COLOURCHECKERS_SDS	Aggregated *Colour Checkers* spectral distributions.

colour.COLOURCHECKERS

colour.**COLOURCHECKERS = CaseInsensitiveMapping({'ColorChecker 1976': ..., 'ColorChecker 2005': ..., 'Bab**

Aggregated *Colour Checkers* chromaticity coordinates.

References

[Bab12b], [Bab12a], [XR15]

COLOURCHECKERS [CaseInsensitiveMapping] **{'ColorChecker 1976', 'ColorChecker 2005', 'BabelColor Average', 'ColorChecker24 - Before November 2014', 'ColorChecker24 - After November 2014'}**

Aliases:

- 'babel_average': 'BabelColor Average'
- 'cc2005': 'ColorChecker 2005'
- 'ccb2014': 'ColorChecker24 - Before November 2014'
- 'cca2014': 'ColorChecker24 - After November 2014'

colour.COLOURCHECKERS_SDS

colour.**COLOURCHECKERS_SDS = CaseInsensitiveMapping({'BabelColor Average': ..., 'ColorChecker N Ohta': ..**
Aggregated *Colour Checkers* spectral distributions.

References

[Oht97], [Bab12b], [Bab12a], [MunsellCSciencea]

COLOURCHECKERS [CaseInsensitiveMapping] **{'BabelColor Average', 'ColorChecker N Ohta'}**

Aliases:

- 'babel_average': 'BabelColor Average'
- 'cc_ohta': 'ColorChecker N Ohta'

Ancillary Objects

colour.characterisation

ColourChecker	*Colour Checker* data.

colour.characterisation.ColourChecker

class colour.characterisation.**ColourChecker**
Colour Checker data.

Parameters

- **name** (unicode) – *Colour Checker* name.
- **data** (OrderedDict) – chromaticity coordinates in *CIE xyY* colourspace.
- **illuminant** (array_like) – *Colour Checker* illuminant chromaticity coordinates.

Create new instance of ColourChecker(name, data, illuminant)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

data	Alias for field number 1
illuminant	Alias for field number 2
name	Alias for field number 0

Cameras

colour.characterisation

RGB_SpectralSensitivities([data, domain, labels])	Implements support for a camera *RGB* spectral sensitivities.

colour.characterisation.RGB_SpectralSensitivities

class colour.characterisation.**RGB_SpectralSensitivities**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for a camera *RGB* spectral sensitivities.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to colour.characterisation.RGB_SpectralSensitivities.labels attribute value.

Dataset

colour

CAMERAS_RGB_SPECTRAL_SENSITIVITIES	Cameras *RGB* spectral sensitivities.

colour.CAMERAS_RGB_SPECTRAL_SENSITIVITIES

colour.**CAMERAS_RGB_SPECTRAL_SENSITIVITIES = CaseInsensitiveMapping({'Nikon 5100 (NPL)': ..., 'Sigma SDMer**
Cameras *RGB* spectral sensitivities.

References

[DFGM15]

CAMERAS_RGB_SPECTRAL_SENSITIVITIES [CaseInsensitiveMapping] **{Nikon 5100 (NPL), Sigma SDMerill (NPL)}**

Displays

colour.characterisation

RGB_DisplayPrimaries([data, domain, labels])	Implements support for a *RGB* display (such as a *CRT* or *LCD*) primaries multi-spectral distributions.

colour.characterisation.RGB_DisplayPrimaries

class colour.characterisation.**RGB_DisplayPrimaries**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)
Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for a *RGB* display (such as a *CRT* or *LCD*) primaries multi-spectral distributions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to colour.characterisation.RGB_DisplayPrimaries.labels attribute value.

Dataset

`colour`

DISPLAYS_RGB_PRIMARIES	Displays *RGB* primaries multi-spectral distributions.

colour.DISPLAYS_RGB_PRIMARIES

`colour.`**`DISPLAYS_RGB_PRIMARIES = CaseInsensitiveMapping({'Typical CRT Brainard 1997': ..., 'Apple Studio D`**

Displays *RGB* primaries multi-spectral distributions.

References

[FW98], [Mac10]

DISPLAYS_RGB_PRIMARIES [CaseInsensitiveMapping] **{Apple Studio Display, Typical CRT Brainard 1997}**

Colorimetry

- *Spectral Data Structure*
- *Spectral Data Generation*
- *Conversion to Tristimulus Values*
  - *ASTM E308-15*
  - *Integration*
- *Spectral Bandpass Dependence Correction*
  - *Stearns and Stearns (1988)*
- *Colour Matching Functions*
- *Colour Matching Functions Transformations*
- *Illuminants and Light Sources*
- *Dominant Wavelength and Purity*
- *Luminous Efficiency Functions*
- *Lightness Computation*
  - *Glasser, Mckinney, Reilly and Schnelle (1958)*
  - *Wyszecki (1963)*
  - *CIE 1976*
  - *Fairchild and Wyble (2010)*
  - *Fairchild and Chen (2011)*
- *Luminance Computation*
  - *Newhall, Nickerson and Judd (1943)*
  - *CIE 1976*
  - *ASTM D1535-08e1*

- *Fairchild and Wyble (2010)*
- *Fairchild and Chen (2011)*

- *Whiteness Computation*
  - *Berger (1959)*
  - *Taube (1960)*
  - *Stensby (1968)*
  - *ASTM E313*
  - *Ganz and Griesser (1979)*
  - *CIE 2004*
- *Yellowness Computation*
  - *ASTM D1925*
  - *ASTM E313*

Spectral Data Structure

colour

SpectralShape([start, end, interval])	Defines the base object for spectral distribution shape.
SpectralDistribution([data, domain])	Defines the spectral distribution: the base object for spectral computations.
MultiSpectralDistributions([data, domain, ...])	Defines the multi-spectral distributions: the base object for multi spectral computations.

colour.SpectralShape

class colour.**SpectralShape**(*start=None*, *end=None*, *interval=None*)

Bases: object

Defines the base object for spectral distribution shape.

Parameters

- **start** (numeric, optional) – Wavelength λ_i range start in nm.
- **end** (numeric, optional) – Wavelength λ_i range end in nm.
- **interval** (numeric, optional) – Wavelength λ_i range interval.

start

end

interval

boundaries

__str__()

__repr__()

__hash__()

__iter__()

__contains__()

__len__()

__eq__()

__ne__()

range()

Examples

```
>>> SpectralShape(360, 830, 1)
SpectralShape(360, 830, 1)
```

property boundaries
Getter and setter property for the spectral shape boundaries.

Parameters **value** (array_like) – Value to set the spectral shape boundaries with.

Returns Spectral shape boundaries.

Return type tuple

property end
Getter and setter property for the spectral shape end.

Parameters **value** (numeric) – Value to set the spectral shape end with.

Returns Spectral shape end.

Return type numeric

property interval
Getter and setter property for the spectral shape interval.

Parameters **value** (numeric) – Value to set the spectral shape interval with.

Returns Spectral shape interval.

Return type numeric

range(*dtype=<class 'numpy.float64'>*)
Returns an iterable range for the spectral shape.

Parameters **dtype** (type) – Data type used to generate the range.

Returns Iterable range for the spectral distribution shape

Return type ndarray

Raises **RuntimeError** – If one of spectral shape *start*, *end* or *interval* attributes is not defined.

Examples

```
>>> SpectralShape(0, 10, 0.1).range()
array([  0. ,   0.1,   0.2,   0.3,   0.4,   0.5,   0.6,   0.7,   0.8,
         0.9,   1. ,   1.1,   1.2,   1.3,   1.4,   1.5,   1.6,   1.7,
         1.8,   1.9,   2. ,   2.1,   2.2,   2.3,   2.4,   2.5,   2.6,
         2.7,   2.8,   2.9,   3. ,   3.1,   3.2,   3.3,   3.4,   3.5,
         3.6,   3.7,   3.8,   3.9,   4. ,   4.1,   4.2,   4.3,   4.4,
         4.5,   4.6,   4.7,   4.8,   4.9,   5. ,   5.1,   5.2,   5.3,
         5.4,   5.5,   5.6,   5.7,   5.8,   5.9,   6. ,   6.1,   6.2,
         6.3,   6.4,   6.5,   6.6,   6.7,   6.8,   6.9,   7. ,   7.1,
         7.2,   7.3,   7.4,   7.5,   7.6,   7.7,   7.8,   7.9,   8. ,
         8.1,   8.2,   8.3,   8.4,   8.5,   8.6,   8.7,   8.8,   8.9,
```

(continues on next page)

(continued from previous page)

```
        9. ,  9.1,  9.2,  9.3,  9.4,  9.5,  9.6,  9.7,  9.8,
        9.9,  10. ])
```

property start

Getter and setter property for the spectral shape start.

Parameters **value** (numeric) – Value to set the spectral shape start with.

Returns Spectral shape start.

Return type numeric

colour.SpectralDistribution

class colour.**SpectralDistribution**(*data=None*, *domain=None*, ***kwargs*)

Bases: colour.continuous.signal.Signal

Defines the spectral distribution: the base object for spectral computations.

The spectral distribution will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Parameters

- **data** (Series or Signal, SpectralDistribution or array_like or dict_like, optional) – Data to be stored in the spectral distribution.
- **domain** (array_like, optional) – Values to initialise the colour.SpectralDistribution.wavelength attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.SpectralDistribution.wavelength attribute.

Other Parameters

- **name** (*unicode, optional*) – Spectral distribution name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function.
- **strict_name** (*unicode, optional*) – Spectral distribution name for figures, default to colour.SpectralDistribution.name attribute value.

strict_name

wavelengths

values

shape

__init__()

extrapolate()

interpolate()

align()

trim()

normalise()

References

[CIET13805a], [CIET13805c], [CIET14804h]

Examples

Instantiating a spectral distribution with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data)
SpectralDistribution([[ 500.    ,    0.0651],
                      [ 520.    ,    0.0705],
                      [ 540.    ,    0.0772],
                      [ 560.    ,    0.087 ],
                      [ 580.    ,    0.1128],
                      [ 600.    ,    0.136 ]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> with numpy_print_options(suppress=True):
...     SpectralDistribution(data)
SpectralDistribution([[ 500.     ,    0.0651 ],
                      [ 510.     ,    0.31416],
                      [ 520.     ,    0.0705 ],
                      [ 540.     ,    0.0772 ],
                      [ 560.     ,    0.087  ],
                      [ 580.     ,    0.1128 ],
                      [ 600.     ,    0.136  ]],
                     interpolator=CubicSplineInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
```

align(*shape*, *interpolator=None*, *interpolator_args=None*, *extrapolator=None*, *extrapolator_args=None*)

Aligns the spectral distribution in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.

Parameters

- **shape** (SpectralShape) – Spectral shape used for alignment.

- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.

Returns Aligned spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.align(SpectralShape(505, 565, 1)))
...
[[ 505.            0.0663929...]
 [ 506.            0.0666509...]
 [ 507.            0.0669069...]
 [ 508.            0.0671613...]
 [ 509.            0.0674150...]
 [ 510.            0.0676692...]
 [ 511.            0.0679253...]
 [ 512.            0.0681848...]
 [ 513.            0.0684491...]
 [ 514.            0.0687197...]
 [ 515.            0.0689975...]
 [ 516.            0.0692832...]
 [ 517.            0.0695771...]
 [ 518.            0.0698787...]
 [ 519.            0.0701870...]
 [ 520.            0.0705   ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
 [ 524.            0.0717789...]
 [ 525.            0.0721063...]
 [ 526.            0.0724367...]
 [ 527.            0.0727698...]
 [ 528.            0.0731051...]
 [ 529.            0.0734423...]
 [ 530.            0.0737808...]
 [ 531.            0.0741203...]
 [ 532.            0.0744603...]
 [ 533.            0.0748006...]
 [ 534.            0.0751409...]
 [ 535.            0.0754813...]
 [ 536.            0.0758220...]
```

(continues on next page)

(continued from previous page)

```
[ 537.            0.0761633...]
[ 538.            0.0765060...]
[ 539.            0.0768511...]
[ 540.            0.0772   ...]
[ 541.            0.0775527...]
[ 542.            0.0779042...]
[ 543.            0.0782507...]
[ 544.            0.0785908...]
[ 545.            0.0789255...]
[ 546.            0.0792576...]
[ 547.            0.0795917...]
[ 548.            0.0799334...]
[ 549.            0.0802895...]
[ 550.            0.0806671...]
[ 551.            0.0810740...]
[ 552.            0.0815176...]
[ 553.            0.0820049...]
[ 554.            0.0825423...]
[ 555.            0.0831351...]
[ 556.            0.0837873...]
[ 557.            0.0845010...]
[ 558.            0.0852763...]
[ 559.            0.0861110...]
[ 560.            0.087    ...]
[ 561.            0.0879383...]
[ 562.            0.0889300...]
[ 563.            0.0899793...]
[ 564.            0.0910876...]
[ 565.            0.0922541...]]
```

extrapolate(*shape*, *extrapolator=None*, *extrapolator_args=None*)

Extrapolates the spectral distribution in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.

Parameters

- **shape** (SpectralShape) – Spectral shape used for extrapolation.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.

Returns Extrapolated spectral distribution.

Return type *SpectralDistribution*

References

[CIET13805c], [CIET14804h]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data)
>>> sd.extrapolate(SpectralShape(400, 700)).shape
SpectralShape(400.0, 700.0, 20.0)
>>> with numpy_print_options(suppress=True):
...     print(sd)
[[ 400.         0.0651]
 [ 420.         0.0651]
 [ 440.         0.0651]
 [ 460.         0.0651]
 [ 480.         0.0651]
 [ 500.         0.0651]
 [ 520.         0.0705]
 [ 540.         0.0772]
 [ 560.         0.087 ]
 [ 580.         0.1128]
 [ 600.         0.136 ]
 [ 620.         0.136 ]
 [ 640.         0.136 ]
 [ 660.         0.136 ]
 [ 680.         0.136 ]
 [ 700.         0.136 ]]
```

interpolate(*shape*, *interpolator=None*, *interpolator_args=None*)

Interpolates the spectral distribution in-place according to *CIE 167:2005* recommendation or given interpolation arguments.

Parameters

- **shape** (SpectralShape, optional) – Spectral shape used for interpolation.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function.

Returns Interpolated spectral distribution.

Return type *SpectralDistribution*

Notes

- Interpolation will be performed over boundaries range, if you need to extend the range of the spectral distribution use the `colour.SpectralDistribution.extrapolate()` or `colour.SpectralDistribution.align()` methods.

Warning:

- *Cubic Spline* interpolator requires at least 3 wavelengths λ_n for interpolation.
- *Sprague (1880)* interpolator requires at least 6 wavelengths λ_n for interpolation.

References

[CIET13805a]

Examples

Spectral distribution with a uniformly spaced independent variable uses *Sprague (1880)* interpolation:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.interpolate(SpectralShape(interval=1)))
...
[[ 500.            0.0651   ...]
 [ 501.            0.0653522...]
 [ 502.            0.0656105...]
 [ 503.            0.0658715...]
 [ 504.            0.0661328...]
 [ 505.            0.0663929...]
 [ 506.            0.0666509...]
 [ 507.            0.0669069...]
 [ 508.            0.0671613...]
 [ 509.            0.0674150...]
 [ 510.            0.0676692...]
 [ 511.            0.0679253...]
 [ 512.            0.0681848...]
 [ 513.            0.0684491...]
 [ 514.            0.0687197...]
 [ 515.            0.0689975...]
 [ 516.            0.0692832...]
 [ 517.            0.0695771...]
 [ 518.            0.0698787...]
 [ 519.            0.0701870...]
 [ 520.            0.0705   ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
```

(continues on next page)

(continued from previous page)

```
[ 524.            0.0717789...]
[ 525.            0.0721063...]
[ 526.            0.0724367...]
[ 527.            0.0727698...]
[ 528.            0.0731051...]
[ 529.            0.0734423...]
[ 530.            0.0737808...]
[ 531.            0.0741203...]
[ 532.            0.0744603...]
[ 533.            0.0748006...]
[ 534.            0.0751409...]
[ 535.            0.0754813...]
[ 536.            0.0758220...]
[ 537.            0.0761633...]
[ 538.            0.0765060...]
[ 539.            0.0768511...]
[ 540.            0.0772   ...]
[ 541.            0.0775527...]
[ 542.            0.0779042...]
[ 543.            0.0782507...]
[ 544.            0.0785908...]
[ 545.            0.0789255...]
[ 546.            0.0792576...]
[ 547.            0.0795917...]
[ 548.            0.0799334...]
[ 549.            0.0802895...]
[ 550.            0.0806671...]
[ 551.            0.0810740...]
[ 552.            0.0815176...]
[ 553.            0.0820049...]
[ 554.            0.0825423...]
[ 555.            0.0831351...]
[ 556.            0.0837873...]
[ 557.            0.0845010...]
[ 558.            0.0852763...]
[ 559.            0.0861110...]
[ 560.            0.087    ...]
[ 561.            0.0879383...]
[ 562.            0.0889300...]
[ 563.            0.0899793...]
[ 564.            0.0910876...]
[ 565.            0.0922541...]
[ 566.            0.0934760...]
[ 567.            0.0947487...]
[ 568.            0.0960663...]
[ 569.            0.0974220...]
[ 570.            0.0988081...]
[ 571.            0.1002166...]
[ 572.            0.1016394...]
[ 573.            0.1030687...]
[ 574.            0.1044972...]
[ 575.            0.1059186...]
[ 576.            0.1073277...]
[ 577.            0.1087210...]
[ 578.            0.1100968...]
[ 579.            0.1114554...]
[ 580.            0.1128   ...]
[ 581.            0.1141333...]
[ 582.            0.1154495...]
[ 583.            0.1167424...]
[ 584.            0.1180082...]
```

(continues on next page)

(continued from previous page)

```
 [ 585.            0.1192452...]
 [ 586.            0.1204536...]
 [ 587.            0.1216348...]
 [ 588.            0.1227915...]
 [ 589.            0.1239274...]
 [ 590.            0.1250465...]
 [ 591.            0.1261531...]
 [ 592.            0.1272517...]
 [ 593.            0.1283460...]
 [ 594.            0.1294393...]
 [ 595.            0.1305340...]
 [ 596.            0.1316310...]
 [ 597.            0.1327297...]
 [ 598.            0.1338277...]
 [ 599.            0.1349201...]
 [ 600.            0.136    ...]]
```

Spectral distribution with a no-uniformly spaced independent variable uses *Cubic Spline* interpolation:

```
>>> sd = SpectralDistribution(data)
>>> sd[510] = np.pi / 10
>>> with numpy_print_options(suppress=True):
...     print(sd.interpolate(SpectralShape(interval=1)))
...
[[ 500.            0.0651   ...]
 [ 501.            0.1365202...]
 [ 502.            0.1953263...]
 [ 503.            0.2423724...]
 [ 504.            0.2785126...]
 [ 505.            0.3046010...]
 [ 506.            0.3214916...]
 [ 507.            0.3300387...]
 [ 508.            0.3310962...]
 [ 509.            0.3255184...]
 [ 510.            0.3141592...]
 [ 511.            0.2978729...]
 [ 512.            0.2775135...]
 [ 513.            0.2539351...]
 [ 514.            0.2279918...]
 [ 515.            0.2005378...]
 [ 516.            0.1724271...]
 [ 517.            0.1445139...]
 [ 518.            0.1176522...]
 [ 519.            0.0926962...]
 [ 520.            0.0705   ...]
 [ 521.            0.0517370...]
 [ 522.            0.0363589...]
 [ 523.            0.0241365...]
 [ 524.            0.0148407...]
 [ 525.            0.0082424...]
 [ 526.            0.0041126...]
 [ 527.            0.0022222...]
 [ 528.            0.0023421...]
 [ 529.            0.0042433...]
 [ 530.            0.0076966...]
 [ 531.            0.0124729...]
 [ 532.            0.0183432...]
 [ 533.            0.0250785...]
 [ 534.            0.0324496...]
 [ 535.            0.0402274...]
```

(continues on next page)

(continued from previous page)

```
[ 536.            0.0481829...]
[ 537.            0.0560870...]
[ 538.            0.0637106...]
[ 539.            0.0708246...]
[ 540.            0.0772   ...]
[ 541.            0.0826564...]
[ 542.            0.0872086...]
[ 543.            0.0909203...]
[ 544.            0.0938549...]
[ 545.            0.0960760...]
[ 546.            0.0976472...]
[ 547.            0.0986321...]
[ 548.            0.0990942...]
[ 549.            0.0990971...]
[ 550.            0.0987043...]
[ 551.            0.0979794...]
[ 552.            0.0969861...]
[ 553.            0.0957877...]
[ 554.            0.0944480...]
[ 555.            0.0930304...]
[ 556.            0.0915986...]
[ 557.            0.0902161...]
[ 558.            0.0889464...]
[ 559.            0.0878532...]
[ 560.            0.087    ...]
[ 561.            0.0864371...]
[ 562.            0.0861623...]
[ 563.            0.0861600...]
[ 564.            0.0864148...]
[ 565.            0.0869112...]
[ 566.            0.0876336...]
[ 567.            0.0885665...]
[ 568.            0.0896945...]
[ 569.            0.0910020...]
[ 570.            0.0924735...]
[ 571.            0.0940936...]
[ 572.            0.0958467...]
[ 573.            0.0977173...]
[ 574.            0.0996899...]
[ 575.            0.1017491...]
[ 576.            0.1038792...]
[ 577.            0.1060649...]
[ 578.            0.1082906...]
[ 579.            0.1105408...]
[ 580.            0.1128   ...]
[ 581.            0.1150526...]
[ 582.            0.1172833...]
[ 583.            0.1194765...]
[ 584.            0.1216167...]
[ 585.            0.1236884...]
[ 586.            0.1256760...]
[ 587.            0.1275641...]
[ 588.            0.1293373...]
[ 589.            0.1309798...]
[ 590.            0.1324764...]
[ 591.            0.1338114...]
[ 592.            0.1349694...]
[ 593.            0.1359349...]
[ 594.            0.1366923...]
[ 595.            0.1372262...]
[ 596.            0.1375211...]
```

(continues on next page)

(continued from previous page)

```
 [ 597.            0.1375614...]
 [ 598.            0.1373316...]
 [ 599.            0.1368163...]
 [ 600.            0.136    ...]]
```

normalise(*factor=1*)

Normalises the spectral distribution using given normalization factor.

Parameters **factor** (numeric, optional) – Normalization factor

Returns Normalised spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data)
>>> with numpy_print_options(suppress=True):
...     print(sd.normalise())
[[ 500.            0.4786764...]
 [ 520.            0.5183823...]
 [ 540.            0.5676470...]
 [ 560.            0.6397058...]
 [ 580.            0.8294117...]
 [ 600.            1.       ...]]
```

property shape

Getter and setter property for the spectral distribution shape.

Returns Spectral distribution shape.

Return type *SpectralShape*

Notes

- A spectral distribution with a non-uniformly spaced independent variable have multiple intervals, in that case colour.SpectralDistribution.shape attribute returns the *minimum* interval size.

Warning: colour.SpectralDistribution.shape attribute is read only.

Examples

Shape of a spectral distribution with a uniformly spaced independent variable:

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> SpectralDistribution(data).shape
SpectralShape(500.0, 600.0, 20.0)
```

Shape of a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[510] = 0.31416
>>> SpectralDistribution(data).shape
SpectralShape(500.0, 600.0, 10.0)
```

property strict_name

Getter and setter property for the spectral distribution strict name.

Parameters **value** (unicode) – Value to set the spectral distribution strict name with.

Returns Spectral distribution strict name.

Return type unicode

trim(*shape*)

Trims the spectral distribution wavelengths to given spectral shape.

Parameters **shape** (SpectralShape) – Spectral shape used for trimming.

Returns Trimmed spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data)
>>> sd = sd.interpolate(SpectralShape(interval=1))
>>> with numpy_print_options(suppress=True):
...     print(sd.trim(SpectralShape(520, 580, 5)))
...
[[ 520.            0.0705   ...]
 [ 521.            0.0708155...]
 [ 522.            0.0711336...]
 [ 523.            0.0714547...]
 [ 524.            0.0717789...]
 [ 525.            0.0721063...]
 [ 526.            0.0724367...]
 [ 527.            0.0727698...]
```

(continues on next page)

(continued from previous page)

```
[ 528.           0.0731051...]
[ 529.           0.0734423...]
[ 530.           0.0737808...]
[ 531.           0.0741203...]
[ 532.           0.0744603...]
[ 533.           0.0748006...]
[ 534.           0.0751409...]
[ 535.           0.0754813...]
[ 536.           0.0758220...]
[ 537.           0.0761633...]
[ 538.           0.0765060...]
[ 539.           0.0768511...]
[ 540.           0.0772   ...]
[ 541.           0.0775527...]
[ 542.           0.0779042...]
[ 543.           0.0782507...]
[ 544.           0.0785908...]
[ 545.           0.0789255...]
[ 546.           0.0792576...]
[ 547.           0.0795917...]
[ 548.           0.0799334...]
[ 549.           0.0802895...]
[ 550.           0.0806671...]
[ 551.           0.0810740...]
[ 552.           0.0815176...]
[ 553.           0.0820049...]
[ 554.           0.0825423...]
[ 555.           0.0831351...]
[ 556.           0.0837873...]
[ 557.           0.0845010...]
[ 558.           0.0852763...]
[ 559.           0.0861110...]
[ 560.           0.087    ...]
[ 561.           0.0879383...]
[ 562.           0.0889300...]
[ 563.           0.0899793...]
[ 564.           0.0910876...]
[ 565.           0.0922541...]
[ 566.           0.0934760...]
[ 567.           0.0947487...]
[ 568.           0.0960663...]
[ 569.           0.0974220...]
[ 570.           0.0988081...]
[ 571.           0.1002166...]
[ 572.           0.1016394...]
[ 573.           0.1030687...]
[ 574.           0.1044972...]
[ 575.           0.1059186...]
[ 576.           0.1073277...]
[ 577.           0.1087210...]
[ 578.           0.1100968...]
[ 579.           0.1114554...]
[ 580.           0.1128   ...]]
```

property values

Getter and setter property for the spectral distribution values.

Parameters **value** (array_like) – Value to set the spectral distribution wavelengths values with.

Returns Spectral distribution values.

Return type ndarray

property wavelengths
Getter and setter property for the spectral distribution wavelengths λ_n.

Parameters **value** (array_like) – Value to set the spectral distribution wavelengths λ_n with.

Returns Spectral distribution wavelengths λ_n.

Return type ndarray

colour.MultiSpectralDistributions

class colour.**MultiSpectralDistributions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)
Bases: colour.continuous.multi_signals.MultiSignals

Defines the multi-spectral distributions: the base object for multi spectral computations. It is used to model colour matching functions, display primaries, camera sensitivities, etc...

The multi-spectral distributions will be initialised according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable. Extrapolation is performed according to *CIE 167:2005* recommendation.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to colour.MultiSpectralDistributions.labels attribute value.

strict_name

strict_labels

wavelengths

values

shape

extrapolate()

interpolate()

align()

trim()

normalise()

to_sds()

References

[CIET13805a], [CIET13805c], [CIET14804h]

Examples

Instantiating the multi-spectral distributions with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> labels = ('x_bar', 'y_bar', 'z_bar')
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistributions(data, labels=labels)
...
MultiSpectral...([[ 500.     ,    0.0049 ,    0.323  ,    0.272  ],
                 ...  [ 510.     ,    0.0093 ,    0.503  ,    0.1582 ],
                 ...  [ 520.     ,    0.06327,    0.71   ,    0.07825],
                 ...  [ 530.     ,    0.1655 ,    0.862  ,    0.04216],
                 ...  [ 540.     ,    0.2904 ,    0.954  ,    0.0203 ],
                 ...  [ 550.     ,    0.43345,    0.99495,    0.00875],
                 ...  [ 560.     ,    0.5945 ,    0.995  ,    0.0039 ]],
                 ... labels=[...'x_bar', ...'y_bar', ...'z_bar'],
                 ... interpolator=SpragueInterpolator,
                 ... interpolator_args={},
                 ... extrapolator=Extrapolator,
                 ... extrapolator_args={...})
```

Instantiating a spectral distribution with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> with numpy_print_options(suppress=True):
...     MultiSpectralDistributions(data, labels=labels)
...
MultiSpectral...([[ 500.     ,    0.0049 ,    0.323  ,    0.272  ],
                 ...  [ 510.     ,    0.0093 ,    0.503  ,    0.1582 ],
                 ...  [ 511.     ,    0.00314,    0.31416,    0.03142],
                 ...  [ 520.     ,    0.06327,    0.71   ,    0.07825],
                 ...  [ 530.     ,    0.1655 ,    0.862  ,    0.04216],
                 ...  [ 540.     ,    0.2904 ,    0.954  ,    0.0203 ],
                 ...  [ 550.     ,    0.43345,    0.99495,    0.00875],
                 ...  [ 560.     ,    0.5945 ,    0.995  ,    0.0039 ]],
                 ... labels=[...'x_bar', ...'y_bar', ...'z_bar'],
                 ... interpolator=CubicSplineInterpolator,
                 ... interpolator_args={},
                 ... extrapolator=Extrapolator,
                 ... extrapolator_args={...})
```

align(*shape*, *interpolator=None*, *interpolator_args=None*, *extrapolator=None*, *extrapolator_args=None*)

Aligns the multi-spectral distributions in-place to given spectral shape: Interpolates first then extrapolates to fit the given range.

Parameters

- **shape** (SpectralShape) – Spectral shape used for alignment.
- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.

Returns Aligned multi-spectral distributions.

Return type *MultiSpectralDistributions*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.align(SpectralShape(505, 565, 1)))
...
[[ 505.            0.0031582...    0.4091067...    0.2126801...]
 [ 506.            0.0035019...    0.4268629...    0.2012748...]
 [ 507.            0.0042365...    0.4450668...    0.1900968...]
 [ 508.            0.0054192...    0.4638181...    0.1791709...]
 [ 509.            0.0070965...    0.4831505...    0.1685260...]
 [ 510.            0.0093   ...    0.503    ...    0.1582   ...]
 [ 511.            0.0120562...    0.5232543...    0.1482365...]
 [ 512.            0.0154137...    0.5439717...    0.1386625...]
 [ 513.            0.0193991...    0.565139 ...    0.1294993...]
 [ 514.            0.0240112...    0.5866255...    0.1207676...]
 [ 515.            0.0292289...    0.6082226...    0.1124864...]
 [ 516.            0.0350192...    0.6296821...    0.1046717...]
 [ 517.            0.0413448...    0.6507558...    0.0973361...]
 [ 518.            0.0481727...    0.6712346...    0.0904871...]
 [ 519.            0.0554816...    0.6909873...    0.0841267...]
 [ 520.            0.06327  ...    0.71     ...    0.07825  ...]
 [ 521.            0.0715642...    0.7283456...    0.0728614...]
 [ 522.            0.0803970...    0.7459679...    0.0680051...]
 [ 523.            0.0897629...    0.7628184...    0.0636823...]
 [ 524.            0.0996227...    0.7789004...    0.0598449...]
 [ 525.            0.1099142...    0.7942533...    0.0564111...]
 [ 526.            0.1205637...    0.8089368...    0.0532822...]
```

(continues on next page)

(continued from previous page)

```
[ 527.          0.1314973...  0.8230153...  0.0503588...]
[ 528.          0.1426523...  0.8365417...  0.0475571...]
[ 529.          0.1539887...  0.8495422...  0.0448253...]
[ 530.          0.1655   ...  0.862    ...  0.04216  ...]
[ 531.          0.1772055...  0.8738585...  0.0395936...]
[ 532.          0.1890877...  0.8850940...  0.0371046...]
[ 533.          0.2011304...  0.8957073...  0.0346733...]
[ 534.          0.2133310...  0.9057092...  0.0323006...]
[ 535.          0.2256968...  0.9151181...  0.0300011...]
[ 536.          0.2382403...  0.9239560...  0.0277974...]
[ 537.          0.2509754...  0.9322459...  0.0257131...]
[ 538.          0.2639130...  0.9400080...  0.0237668...]
[ 539.          0.2770569...  0.9472574...  0.0219659...]
[ 540.          0.2904   ...  0.954    ...  0.0203   ...]
[ 541.          0.3039194...  0.9602409...  0.0187414...]
[ 542.          0.3175893...  0.9660106...  0.0172748...]
[ 543.          0.3314022...  0.9713260...  0.0158947...]
[ 544.          0.3453666...  0.9761850...  0.0146001...]
[ 545.          0.3595019...  0.9805731...  0.0133933...]
[ 546.          0.3738324...  0.9844703...  0.0122777...]
[ 547.          0.3883818...  0.9878583...  0.0112562...]
[ 548.          0.4031674...  0.9907270...  0.0103302...]
[ 549.          0.4181943...  0.9930817...  0.0094972...]
[ 550.          0.43345  ...  0.99495  ...  0.00875  ...]
[ 551.          0.4489082...  0.9963738...  0.0080748...]
[ 552.          0.4645599...  0.9973682...  0.0074580...]
[ 553.          0.4803950...  0.9979568...  0.0068902...]
[ 554.          0.4963962...  0.9981802...  0.0063660...]
[ 555.          0.5125410...  0.9980910...  0.0058818...]
[ 556.          0.5288034...  0.9977488...  0.0054349...]
[ 557.          0.5451560...  0.9972150...  0.0050216...]
[ 558.          0.5615719...  0.9965479...  0.0046357...]
[ 559.          0.5780267...  0.9957974...  0.0042671...]
[ 560.          0.5945   ...  0.995    ...  0.0039   ...]
[ 561.          0.5945   ...  0.995    ...  0.0039   ...]
[ 562.          0.5945   ...  0.995    ...  0.0039   ...]
[ 563.          0.5945   ...  0.995    ...  0.0039   ...]
[ 564.          0.5945   ...  0.995    ...  0.0039   ...]
[ 565.          0.5945   ...  0.995    ...  0.0039   ...]]
```

extrapolate(*shape*, *extrapolator=None*, *extrapolator_args=None*)

Extrapolates the multi-spectral distributions in-place according to *CIE 15:2004* and *CIE 167:2005* recommendations or given extrapolation arguments.

Parameters

- **shape** (SpectralShape) – Spectral shape used for extrapolation.
- **extrapolator** (object, optional) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (dict_like, optional) – Arguments to use when instantiating the extrapolating function.

Returns Extrapolated multi-spectral distributions.

Return type *MultiSpectralDistributions*

References

[CIET13805c], [CIET14804h]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> msds = MultiSpectralDistributions(data)
>>> msds.extrapolate(SpectralShape(400, 700)).shape
SpectralShape(400.0, 700.0, 10.0)
>>> with numpy_print_options(suppress=True):
...     print(msds)
[[ 400.         0.0049     0.323      0.272  ]
 [ 410.         0.0049     0.323      0.272  ]
 [ 420.         0.0049     0.323      0.272  ]
 [ 430.         0.0049     0.323      0.272  ]
 [ 440.         0.0049     0.323      0.272  ]
 [ 450.         0.0049     0.323      0.272  ]
 [ 460.         0.0049     0.323      0.272  ]
 [ 470.         0.0049     0.323      0.272  ]
 [ 480.         0.0049     0.323      0.272  ]
 [ 490.         0.0049     0.323      0.272  ]
 [ 500.         0.0049     0.323      0.272  ]
 [ 510.         0.0093     0.503      0.1582 ]
 [ 520.         0.06327    0.71       0.07825]
 [ 530.         0.1655     0.862      0.04216]
 [ 540.         0.2904     0.954      0.0203 ]
 [ 550.         0.43345    0.99495    0.00875]
 [ 560.         0.5945     0.995      0.0039 ]
 [ 570.         0.5945     0.995      0.0039 ]
 [ 580.         0.5945     0.995      0.0039 ]
 [ 590.         0.5945     0.995      0.0039 ]
 [ 600.         0.5945     0.995      0.0039 ]
 [ 610.         0.5945     0.995      0.0039 ]
 [ 620.         0.5945     0.995      0.0039 ]
 [ 630.         0.5945     0.995      0.0039 ]
 [ 640.         0.5945     0.995      0.0039 ]
 [ 650.         0.5945     0.995      0.0039 ]
 [ 660.         0.5945     0.995      0.0039 ]
 [ 670.         0.5945     0.995      0.0039 ]
 [ 680.         0.5945     0.995      0.0039 ]
 [ 690.         0.5945     0.995      0.0039 ]
 [ 700.         0.5945     0.995      0.0039 ]]
```

interpolate(*shape*, *interpolator=None*, *interpolator_args=None*)

Interpolates the multi-spectral distributions in-place according to *CIE 167:2005* recommendation or given interpolation arguments.

Parameters

- **shape** (SpectralShape, optional) – Spectral shape used for interpolation.

- **interpolator** (object, optional) – Interpolator class type to use as interpolating function.
- **interpolator_args** (dict_like, optional) – Arguments to use when instantiating the interpolating function.

Returns Interpolated multi-spectral distributions.

Return type *MultiSpectralDistributions*

Notes

- See colour.SpectralDistribution.interpolate() method notes section.

Warning: See colour.SpectralDistribution.interpolate() method warning section.

References

[CIET13805a]

Examples

Multi-spectral distributions with a uniformly spaced independent variable uses *Sprague (1880)* interpolation:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.interpolate(SpectralShape(interval=1)))
...
[[ 500.            0.0049    ...    0.323     ...    0.272     ...]
 [ 501.            0.0043252...    0.3400642...    0.2599848...]
 [ 502.            0.0037950...    0.3572165...    0.2479849...]
 [ 503.            0.0033761...    0.3744030...    0.2360688...]
 [ 504.            0.0031397...    0.3916650...    0.2242878...]
 [ 505.            0.0031582...    0.4091067...    0.2126801...]
 [ 506.            0.0035019...    0.4268629...    0.2012748...]
 [ 507.            0.0042365...    0.4450668...    0.1900968...]
 [ 508.            0.0054192...    0.4638181...    0.1791709...]
 [ 509.            0.0070965...    0.4831505...    0.1685260...]
 [ 510.            0.0093    ...    0.503     ...    0.1582    ...]
 [ 511.            0.0120562...    0.5232543...    0.1482365...]
 [ 512.            0.0154137...    0.5439717...    0.1386625...]
 [ 513.            0.0193991...    0.565139  ...    0.1294993...]
 [ 514.            0.0240112...    0.5866255...    0.1207676...]
 [ 515.            0.0292289...    0.6082226...    0.1124864...]
 [ 516.            0.0350192...    0.6296821...    0.1046717...]
 [ 517.            0.0413448...    0.6507558...    0.0973361...]
 [ 518.            0.0481727...    0.6712346...    0.0904871...]
```

(continues on next page)

(continued from previous page)

```
 [ 519.           0.0554816...    0.6909873...    0.0841267...]
 [ 520.           0.06327  ...    0.71     ...    0.07825  ...]
 [ 521.           0.0715642...    0.7283456...    0.0728614...]
 [ 522.           0.0803970...    0.7459679...    0.0680051...]
 [ 523.           0.0897629...    0.7628184...    0.0636823...]
 [ 524.           0.0996227...    0.7789004...    0.0598449...]
 [ 525.           0.1099142...    0.7942533...    0.0564111...]
 [ 526.           0.1205637...    0.8089368...    0.0532822...]
 [ 527.           0.1314973...    0.8230153...    0.0503588...]
 [ 528.           0.1426523...    0.8365417...    0.0475571...]
 [ 529.           0.1539887...    0.8495422...    0.0448253...]
 [ 530.           0.1655   ...    0.862    ...    0.04216  ...]
 [ 531.           0.1772055...    0.8738585...    0.0395936...]
 [ 532.           0.1890877...    0.8850940...    0.0371046...]
 [ 533.           0.2011304...    0.8957073...    0.0346733...]
 [ 534.           0.2133310...    0.9057092...    0.0323006...]
 [ 535.           0.2256968...    0.9151181...    0.0300011...]
 [ 536.           0.2382403...    0.9239560...    0.0277974...]
 [ 537.           0.2509754...    0.9322459...    0.0257131...]
 [ 538.           0.2639130...    0.9400080...    0.0237668...]
 [ 539.           0.2770569...    0.9472574...    0.0219659...]
 [ 540.           0.2904   ...    0.954    ...    0.0203   ...]
 [ 541.           0.3039194...    0.9602409...    0.0187414...]
 [ 542.           0.3175893...    0.9660106...    0.0172748...]
 [ 543.           0.3314022...    0.9713260...    0.0158947...]
 [ 544.           0.3453666...    0.9761850...    0.0146001...]
 [ 545.           0.3595019...    0.9805731...    0.0133933...]
 [ 546.           0.3738324...    0.9844703...    0.0122777...]
 [ 547.           0.3883818...    0.9878583...    0.0112562...]
 [ 548.           0.4031674...    0.9907270...    0.0103302...]
 [ 549.           0.4181943...    0.9930817...    0.0094972...]
 [ 550.           0.43345  ...    0.99495  ...    0.00875  ...]
 [ 551.           0.4489082...    0.9963738...    0.0080748...]
 [ 552.           0.4645599...    0.9973682...    0.0074580...]
 [ 553.           0.4803950...    0.9979568...    0.0068902...]
 [ 554.           0.4963962...    0.9981802...    0.0063660...]
 [ 555.           0.5125410...    0.9980910...    0.0058818...]
 [ 556.           0.5288034...    0.9977488...    0.0054349...]
 [ 557.           0.5451560...    0.9972150...    0.0050216...]
 [ 558.           0.5615719...    0.9965479...    0.0046357...]
 [ 559.           0.5780267...    0.9957974...    0.0042671...]
 [ 560.           0.5945   ...    0.995    ...    0.0039   ...]]
```

Multi-spectral distributions with a non-uniformly spaced independent variable uses *Cubic Spline* interpolation:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.interpolate(SpectralShape(interval=1)))
...
[[ 500.           0.0049   ...    0.323    ...    0.272    ...]
 [ 501.           0.0300110...    0.9455153...    0.5985102...]
 [ 502.           0.0462136...    1.3563103...    0.8066498...]
 [ 503.           0.0547925...    1.5844039...    0.9126502...]
 [ 504.           0.0570325...    1.6588148...    0.9327429...]
 [ 505.           0.0542183...    1.6085619...    0.8831594...]
 [ 506.           0.0476346...    1.4626640...    0.7801312...]
 [ 507.           0.0385662...    1.2501401...    0.6398896...]
 [ 508.           0.0282978...    1.0000089...    0.4786663...]
 [ 509.           0.0181142...    0.7412892...    0.3126925...]
```

(continues on next page)

(continued from previous page)

```
[ 510.           0.0093   ...    0.503    ...    0.1582   ...]
[ 511.           0.00314  ...    0.31416  ...    0.03142  ...]
[ 512.           0.0006228...    0.1970419...   -0.0551709...]
[ 513.           0.0015528...    0.1469341...   -0.1041165...]
[ 514.           0.0054381...    0.1523785...   -0.1217152...]
[ 515.           0.0117869...    0.2019173...   -0.1142659...]
[ 516.           0.0201073...    0.2840925...   -0.0880670...]
[ 517.           0.0299077...    0.3874463...   -0.0494174...]
[ 518.           0.0406961...    0.5005208...   -0.0046156...]
[ 519.           0.0519808...    0.6118579...    0.0400397...]
[ 520.           0.06327  ...    0.71     ...    0.07825  ...]
[ 521.           0.0741690...    0.7859059...    0.1050384...]
[ 522.           0.0846726...    0.8402033...    0.1207164...]
[ 523.           0.0948728...    0.8759363...    0.1269173...]
[ 524.           0.1048614...    0.8961496...    0.1252743...]
[ 525.           0.1147305...    0.9038874...    0.1174207...]
[ 526.           0.1245719...    0.9021942...    0.1049899...]
[ 527.           0.1344776...    0.8941145...    0.0896151...]
[ 528.           0.1445395...    0.8826926...    0.0729296...]
[ 529.           0.1548497...    0.8709729...    0.0565668...]
[ 530.           0.1655   ...    0.862    ...    0.04216  ...]
[ 531.           0.1765618...    0.858179 ...    0.0309976...]
[ 532.           0.1880244...    0.8593588...    0.0229897...]
[ 533.           0.1998566...    0.8647493...    0.0177013...]
[ 534.           0.2120269...    0.8735601...    0.0146975...]
[ 535.           0.2245042...    0.8850011...    0.0135435...]
[ 536.           0.2372572...    0.8982820...    0.0138044...]
[ 537.           0.2502546...    0.9126126...    0.0150454...]
[ 538.           0.2634650...    0.9272026...    0.0168315...]
[ 539.           0.2768572...    0.9412618...    0.0187280...]
[ 540.           0.2904   ...    0.954    ...    0.0203   ...]
[ 541.           0.3040682...    0.9647869...    0.0211987...]
[ 542.           0.3178617...    0.9736329...    0.0214207...]
[ 543.           0.3317865...    0.9807080...    0.0210486...]
[ 544.           0.3458489...    0.9861825...    0.0201650...]
[ 545.           0.3600548...    0.9902267...    0.0188525...]
[ 546.           0.3744103...    0.9930107...    0.0171939...]
[ 547.           0.3889215...    0.9947048...    0.0152716...]
[ 548.           0.4035944...    0.9954792...    0.0131685...]
[ 549.           0.4184352...    0.9955042...    0.0109670...]
[ 550.           0.43345  ...    0.99495  ...    0.00875  ...]
[ 551.           0.4486447...    0.9939867...    0.0065999...]
[ 552.           0.4640255...    0.9927847...    0.0045994...]
[ 553.           0.4795984...    0.9915141...    0.0028313...]
[ 554.           0.4953696...    0.9903452...    0.0013781...]
[ 555.           0.5113451...    0.9894483...    0.0003224...]
[ 556.           0.5275310...    0.9889934...   -0.0002530...]
[ 557.           0.5439334...    0.9891509...   -0.0002656...]
[ 558.           0.5605583...    0.9900910...    0.0003672...]
[ 559.           0.5774118...    0.9919840...    0.0017282...]
[ 560.           0.5945   ...    0.995    ...    0.0039   ...]]
```

normalise(*factor=1*)

Normalises the multi-spectral distributions with given normalization factor.

Parameters **factor** (numeric, optional) – Normalization factor

Returns Normalised multi- spectral distribution.

Return type *MultiSpectralDistributions*

Notes

- The implementation uses the maximum value for each `colour.SpectralDistribution` class instances.

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     print(msds.normalise())
[[ 500.            0.0082422...    0.3246231...    1.        ...]
 [ 510.            0.0156434...    0.5055276...    0.5816176...]
 [ 520.            0.1064255...    0.7135678...    0.2876838...]
 [ 530.            0.2783852...    0.8663316...    0.155     ...]
 [ 540.            0.4884777...    0.9587939...    0.0746323...]
 [ 550.            0.7291000...    0.9999497...    0.0321691...]
 [ 560.            1.        ...   1.        ...   0.0143382...]]
```

property shape

Getter and setter property for the multi-spectral distributions shape.

Returns Multi-spectral distributions shape.

Return type *SpectralShape*

Notes

- Multi-spectral distributions with a non-uniformly spaced independent variable have multiple intervals, in that case `colour.MultiSpectralDistributions.shape` attribute returns the *minimum* interval size.

Warning: `colour.MultiSpectralDistributions.shape` attribute is read only.

Examples

Shape of the multi-spectral distributions with a uniformly spaced independent variable:

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
```

(continues on next page)

(continued from previous page)

```
>>> MultiSpectralDistributions(data).shape
SpectralShape(500.0, 560.0, 10.0)
```

Shape of the multi-spectral distributions with a non-uniformly spaced independent variable:

```
>>> data[511] = (0.00314, 0.31416, 0.03142)
>>> MultiSpectralDistributions(data).shape
SpectralShape(500.0, 560.0, 1.0)
```

property strict_labels

Getter and setter property for the multi-spectral distributions strict labels.

Parameters **value** (array_like) – Value to set the multi-spectral distributions strict labels with.

Returns Multi-spectral distributions strict labels.

Return type array_like

property strict_name

Getter and setter property for the multi-spectral distributions strict name.

Parameters **value** (unicode) – Value to set the multi-spectral distributions strict name with.

Returns Multi-spectral distributions strict name.

Return type unicode

to_sds()

Converts the multi-spectral distributions to a list of spectral distributions and update their name and strict name using the labels and strict labels.

Returns List of spectral distributions.

Return type list

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> msds = MultiSpectralDistributions(data)
>>> with numpy_print_options(suppress=True):
...     for sd in msds.to_sds():
...         print(sd)
[[ 500.          0.0049 ...]
 [ 510.          0.0093 ...]
 [ 520.          0.06327...]
 [ 530.          0.1655 ...]
 [ 540.          0.2904 ...]
 [ 550.          0.43345...]
 [ 560.          0.5945 ...]]
[[ 500.          0.323  ...]
 [ 510.          0.503  ...]
 [ 520.          0.71   ...]
```

(continues on next page)

(continued from previous page)

```
 [ 530.          0.862  ...]
 [ 540.          0.954  ...]
 [ 550.          0.99495...]
 [ 560.          0.995  ...]]
[[ 500.          0.272  ...]
 [ 510.          0.1582 ...]
 [ 520.          0.07825...]
 [ 530.          0.04216...]
 [ 540.          0.0203 ...]
 [ 550.          0.00875...]
 [ 560.          0.0039 ...]]
```

trim(*shape*)

Trims the multi-spectral distributions wavelengths to given shape.

Parameters **shape** (SpectralShape) – Spectral shape used for trimming.

Returns Trimmed multi-spectral distributions.

Return type *MultiSpectralDistributions*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> msds = MultiSpectralDistributions(data)
>>> msds = msds.interpolate(SpectralShape(interval=1))
>>> with numpy_print_options(suppress=True):
...     print(msds.trim(SpectralShape(520, 580, 5)))
...
[[ 520.            0.06327  ...    0.71     ...    0.07825  ...]
 [ 521.            0.0715642...    0.7283456...    0.0728614...]
 [ 522.            0.0803970...    0.7459679...    0.0680051...]
 [ 523.            0.0897629...    0.7628184...    0.0636823...]
 [ 524.            0.0996227...    0.7789004...    0.0598449...]
 [ 525.            0.1099142...    0.7942533...    0.0564111...]
 [ 526.            0.1205637...    0.8089368...    0.0532822...]
 [ 527.            0.1314973...    0.8230153...    0.0503588...]
 [ 528.            0.1426523...    0.8365417...    0.0475571...]
 [ 529.            0.1539887...    0.8495422...    0.0448253...]
 [ 530.            0.1655   ...    0.862    ...    0.04216  ...]
 [ 531.            0.1772055...    0.8738585...    0.0395936...]
 [ 532.            0.1890877...    0.8850940...    0.0371046...]
 [ 533.            0.2011304...    0.8957073...    0.0346733...]
 [ 534.            0.2133310...    0.9057092...    0.0323006...]
 [ 535.            0.2256968...    0.9151181...    0.0300011...]
 [ 536.            0.2382403...    0.9239560...    0.0277974...]
 [ 537.            0.2509754...    0.9322459...    0.0257131...]
 [ 538.            0.2639130...    0.9400080...    0.0237668...]
 [ 539.            0.2770569...    0.9472574...    0.0219659...]
 [ 540.            0.2904   ...    0.954    ...    0.0203   ...]
 [ 541.            0.3039194...    0.9602409...    0.0187414...]
 [ 542.            0.3175893...    0.9660106...    0.0172748...]
```

(continues on next page)

(continued from previous page)

```
[ 543.          0.3314022...    0.9713260...    0.0158947...]
[ 544.          0.3453666...    0.9761850...    0.0146001...]
[ 545.          0.3595019...    0.9805731...    0.0133933...]
[ 546.          0.3738324...    0.9844703...    0.0122777...]
[ 547.          0.3883818...    0.9878583...    0.0112562...]
[ 548.          0.4031674...    0.9907270...    0.0103302...]
[ 549.          0.4181943...    0.9930817...    0.0094972...]
[ 550.          0.43345  ...    0.99495  ...    0.00875  ...]
[ 551.          0.4489082...    0.9963738...    0.0080748...]
[ 552.          0.4645599...    0.9973682...    0.0074580...]
[ 553.          0.4803950...    0.9979568...    0.0068902...]
[ 554.          0.4963962...    0.9981802...    0.0063660...]
[ 555.          0.5125410...    0.9980910...    0.0058818...]
[ 556.          0.5288034...    0.9977488...    0.0054349...]
[ 557.          0.5451560...    0.9972150...    0.0050216...]
[ 558.          0.5615719...    0.9965479...    0.0046357...]
[ 559.          0.5780267...    0.9957974...    0.0042671...]
[ 560.          0.5945   ...    0.995    ...    0.0039   ...]]
```

property values

Getter and setter property for the multi-spectral distributions values.

Parameters **value** (array_like) – Value to set the multi-spectral distributions wavelengths values with.

Returns Multi-spectral distributions values.

Return type ndarray

property wavelengths

Getter and setter property for the multi-spectral distributions wavelengths λ_n.

Parameters **value** (array_like) – Value to set the multi-spectral distributions wavelengths λ_n with.

Returns Multi-spectral distributions wavelengths λ_n.

Return type ndarray

DEFAULT_SPECTRAL_SHAPE	(360, 780, 1).
ASTME308_PRACTISE_SHAPE	(360, 780, 1).

colour.DEFAULT_SPECTRAL_SHAPE

colour.**DEFAULT_SPECTRAL_SHAPE = SpectralShape(360, 780, 1)**

(360, 780, 1).

References

[ASTMInternational15]

ASTME308_PRACTISE_SHAPE : SpectralShape

Type Shape for *ASTM E308-15* practise

colour.ASTME308_PRACTISE_SHAPE

colour.**ASTME308_PRACTISE_SHAPE = SpectralShape(360, 780, 1)**
(360, 780, 1).

References

[ASTMInternational15]

ASTME308_PRACTISE_SHAPE : SpectralShape

Type Shape for *ASTM E308-15* practise

Spectral Data Generation

colour

sd_CIE_standard_illuminant_A([shape])	*CIE Standard Illuminant A* is intended to represent typical, domestic, tungsten-filament lighting.
sd_CIE_illuminant_D_series(xy[, M1_M2_rounding])	Returns the spectral distribution of given *CIE Illuminant D Series* using given *CIE xy* chromaticity coordinates.
sd_blackbody(temperature[, shape, c1, c2, n])	Returns the spectral distribution of the planckian radiator for given temperature $T[K]$.
sd_constant(k[, shape, dtype])	Returns a spectral distribution of given spectral shape filled with constant k values.
sd_ones([shape])	Returns a spectral distribution of given spectral shape filled with ones.
sd_zeros([shape])	Returns a spectral distribution of given spectral shape filled with zeros.
SD_GAUSSIAN_METHODS	Supported gaussian spectral distribution computation methods.
sd_gaussian(mu_peak_wavelength, sigma_fwhm)	Returns a gaussian spectral distribution of given spectral shape using given method.
SD_SINGLE_LED_METHODS	Supported single *LED* spectral distribution computation methods.
sd_single_led(peak_wavelength, fwhm[, ...])	Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to given method.
SD_MULTI_LEDS_METHODS	Supported multi *LED* spectral distribution computation methods.
sd_multi_leds(peak_wavelengths, fwhm[, ...])	Returns a multi *LED* spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to given method.

colour.sd_CIE_standard_illuminant_A

colour.**sd_CIE_standard_illuminant_A**(*shape=SpectralShape(360, 780, 1)*)

CIE Standard Illuminant A is intended to represent typical, domestic, tungsten-filament lighting.

Its spectral distribution is that of a Planckian radiator at a temperature of approximately 2856 K. *CIE Standard Illuminant A* should be used in all applications of colorimetry involving the use of incandescent lighting, unless there are specific reasons for using a different illuminant.

Parameters **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution of the *CIE Standard Illuminant A*.

Returns *CIE Standard Illuminant A*. spectral distribution.

Return type *SpectralDistribution*

References

[CIET14804a]

Examples

```
>>> from colour import SpectralShape
>>> sd_CIE_standard_illuminant_A(SpectralShape(400, 700, 10))
...
SpectralDistribution([[ 400.        ,   14.7080384...],
                      [ 410.        ,   17.6752521...],
                      [ 420.        ,   20.9949572...],
                      [ 430.        ,   24.6709226...],
                      [ 440.        ,   28.7027304...],
                      [ 450.        ,   33.0858929...],
                      [ 460.        ,   37.8120566...],
                      [ 470.        ,   42.8692762...],
                      [ 480.        ,   48.2423431...],
                      [ 490.        ,   53.9131532...],
                      [ 500.        ,   59.8610989...],
                      [ 510.        ,   66.0634727...],
                      [ 520.        ,   72.4958719...],
                      [ 530.        ,   79.1325945...],
                      [ 540.        ,   85.9470183...],
                      [ 550.        ,   92.9119589...],
                      [ 560.        ,  100.       ...],
                      [ 570.        ,  107.1837952...],
                      [ 580.        ,  114.4363383...],
                      [ 590.        ,  121.7312009...],
                      [ 600.        ,  129.0427389...],
                      [ 610.        ,  136.3462674...],
                      [ 620.        ,  143.6182057...],
                      [ 630.        ,  150.8361944...],
                      [ 640.        ,  157.9791857...],
                      [ 650.        ,  165.0275098...],
                      [ 660.        ,  171.9629200...],
                      [ 670.        ,  178.7686175...],
                      [ 680.        ,  185.4292591...],
                      [ 690.        ,  191.9309499...],
                      [ 700.        ,  198.2612232...]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
```

colour.sd_CIE_illuminant_D_series

colour.**sd_CIE_illuminant_D_series**(*xy*, *M1_M2_rounding=True*)

Returns the spectral distribution of given *CIE Illuminant D Series* using given *CIE xy* chromaticity coordinates.

Parameters

- **xy** (array_like) – *CIE xy* chromaticity coordinates.
- **M1_M2_rounding** (bool, optional) – Whether to round $M1$ and $M2$ variables to 3 decimal places in order to yield the internationally agreed values.

Returns *CIE Illuminant D Series* spectral distribution.

Return type *SpectralDistribution*

Notes

- The nominal *CIE xy* chromaticity coordinates which have been computed with colour.temperature.CCT_to_xy_CIE_D() must be given according to *CIE 015:2004* recommendation and thus multiplied by 1.4388 / 1.4380.
- $M1$ **and** $M2$ **variables are rounded to 3 decimal places** according to *CIE 015:2004* recommendation.

References

[CIET14804g], [WS00d]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> from colour.temperature import CCT_to_xy_CIE_D
>>> CCT_D65 = 6500 * 1.4388 / 1.4380
>>> xy = CCT_to_xy_CIE_D(CCT_D65)
>>> with numpy_print_options(suppress=True):
...     sd_CIE_illuminant_D_series(xy)
SpectralDistribution([[ 300.     ,    0.0341...],
                      [ 305.     ,    1.6643...],
                      [ 310.     ,    3.2945...],
                      [ 315.     ,   11.7652...],
                      [ 320.     ,   20.236 ...],
                      [ 325.     ,   28.6447...],
                      [ 330.     ,   37.0535...],
                      [ 335.     ,   38.5011...],
                      [ 340.     ,   39.9488...],
                      [ 345.     ,   42.4302...],
                      [ 350.     ,   44.9117...],
                      [ 355.     ,   45.775 ...],
                      [ 360.     ,   46.6383...],
                      [ 365.     ,   49.3637...],
                      [ 370.     ,   52.0891...],
                      [ 375.     ,   51.0323...],
                      [ 380.     ,   49.9755...],
                      [ 385.     ,   52.3118...],
                      [ 390.     ,   54.6482...],
                      [ 395.     ,   68.7015...],
                      [ 400.     ,   82.7549...],
                      [ 405.     ,   87.1204...],
```

(continues on next page)

(continued from previous page)

```
                       [ 410.      ,   91.486 ...],
                       [ 415.      ,   92.4589...],
                       [ 420.      ,   93.4318...],
                       [ 425.      ,   90.0570...],
                       [ 430.      ,   86.6823...],
                       [ 435.      ,   95.7736...],
                       [ 440.      ,  104.8649...],
                       [ 445.      ,  110.9362...],
                       [ 450.      ,  117.0076...],
                       [ 455.      ,  117.4099...],
                       [ 460.      ,  117.8122...],
                       [ 465.      ,  116.3365...],
                       [ 470.      ,  114.8609...],
                       [ 475.      ,  115.3919...],
                       [ 480.      ,  115.9229...],
                       [ 485.      ,  112.3668...],
                       [ 490.      ,  108.8107...],
                       [ 495.      ,  109.0826...],
                       [ 500.      ,  109.3545...],
                       [ 505.      ,  108.5781...],
                       [ 510.      ,  107.8017...],
                       [ 515.      ,  106.2957...],
                       [ 520.      ,  104.7898...],
                       [ 525.      ,  106.2396...],
                       [ 530.      ,  107.6895...],
                       [ 535.      ,  106.0475...],
                       [ 540.      ,  104.4055...],
                       [ 545.      ,  104.2258...],
                       [ 550.      ,  104.0462...],
                       [ 555.      ,  102.0231...],
                       [ 560.      ,  100.    ...],
                       [ 565.      ,   98.1671...],
                       [ 570.      ,   96.3342...],
                       [ 575.      ,   96.0611...],
                       [ 580.      ,   95.788 ...],
                       [ 585.      ,   92.2368...],
                       [ 590.      ,   88.6856...],
                       [ 595.      ,   89.3459...],
                       [ 600.      ,   90.0062...],
                       [ 605.      ,   89.8026...],
                       [ 610.      ,   89.5991...],
                       [ 615.      ,   88.6489...],
                       [ 620.      ,   87.6987...],
                       [ 625.      ,   85.4936...],
                       [ 630.      ,   83.2886...],
                       [ 635.      ,   83.4939...],
                       [ 640.      ,   83.6992...],
                       [ 645.      ,   81.863 ...],
                       [ 650.      ,   80.0268...],
                       [ 655.      ,   80.1207...],
                       [ 660.      ,   80.2146...],
                       [ 665.      ,   81.2462...],
                       [ 670.      ,   82.2778...],
                       [ 675.      ,   80.281 ...],
                       [ 680.      ,   78.2842...],
                       [ 685.      ,   74.0027...],
                       [ 690.      ,   69.7213...],
                       [ 695.      ,   70.6652...],
                       [ 700.      ,   71.6091...],
                       [ 705.      ,   72.9790...],
                       [ 710.      ,   74.349 ...],
```

(continues on next page)

(continued from previous page)

```
                    [ 715.     ,   67.9765...],
                    [ 720.     ,   61.604 ...],
                    [ 725.     ,   65.7448...],
                    [ 730.     ,   69.8856...],
                    [ 735.     ,   72.4863...],
                    [ 740.     ,   75.087 ...],
                    [ 745.     ,   69.3398...],
                    [ 750.     ,   63.5927...],
                    [ 755.     ,   55.0054...],
                    [ 760.     ,   46.4182...],
                    [ 765.     ,   56.6118...],
                    [ 770.     ,   66.8054...],
                    [ 775.     ,   65.0941...],
                    [ 780.     ,   63.3828...],
                    [ 785.     ,   63.8434...],
                    [ 790.     ,   64.304 ...],
                    [ 795.     ,   61.8779...],
                    [ 800.     ,   59.4519...],
                    [ 805.     ,   55.7054...],
                    [ 810.     ,   51.959 ...],
                    [ 815.     ,   54.6998...],
                    [ 820.     ,   57.4406...],
                    [ 825.     ,   58.8765...],
                    [ 830.     ,   60.3125...]],
                   interpolator=SpragueInterpolator,
                   interpolator_args={},
                   extrapolator=Extrapolator,
                   extrapolator_args={...})
```

colour.sd_blackbody

colour.**sd_blackbody**(*temperature*, *shape=SpectralShape(360, 780, 1)*, *c1=3.741771e-16*, *c2=0.014388*, *n=1*)

Returns the spectral distribution of the planckian radiator for given temperature $T[K]$.

Parameters

- **temperature** (numeric) – Temperature $T[K]$ in kelvin degrees.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution of the planckian radiator.
- **c1** (numeric, optional) – The official value of $c1$ is provided by the Committee on Data for Science and Technology (CODATA) and is $c1 = 3,741771x10.16\ W/m_2$ *(Mohr and Taylor, 2000)*.
- **c2** (numeric, optional) – Since T is measured on the International Temperature Scale, the value of $c2$ used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et aI., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **n** (numeric, optional) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Blackbody spectral distribution.

Return type *SpectralDistribution*

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_blackbody(5000)
SpectralDistribution([[  3.60000000e+02,   6.65427827e+12],
                      [  3.61000000e+02,   6.70960528e+12],
                      [  3.62000000e+02,   6.76482512e+12],
                      [  3.63000000e+02,   6.81993308e+12],
                      [  3.64000000e+02,   6.87492449e+12],
                      [  3.65000000e+02,   6.92979475e+12],
                      [  3.66000000e+02,   6.98453932e+12],
                      [  3.67000000e+02,   7.03915372e+12],
                      [  3.68000000e+02,   7.09363351e+12],
                      [  3.69000000e+02,   7.14797433e+12],
                      [  3.70000000e+02,   7.20217187e+12],
                      [  3.71000000e+02,   7.25622190e+12],
                      [  3.72000000e+02,   7.31012021e+12],
                      [  3.73000000e+02,   7.36386268e+12],
                      [  3.74000000e+02,   7.41744525e+12],
                      [  3.75000000e+02,   7.47086391e+12],
                      [  3.76000000e+02,   7.52411471e+12],
                      [  3.77000000e+02,   7.57719377e+12],
                      [  3.78000000e+02,   7.63009726e+12],
                      [  3.79000000e+02,   7.68282141e+12],
                      [  3.80000000e+02,   7.73536252e+12],
                      [  3.81000000e+02,   7.78771695e+12],
                      [  3.82000000e+02,   7.83988111e+12],
                      [  3.83000000e+02,   7.89185148e+12],
                      [  3.84000000e+02,   7.94362458e+12],
                      [  3.85000000e+02,   7.99519703e+12],
                      [  3.86000000e+02,   8.04656547e+12],
                      [  3.87000000e+02,   8.09772662e+12],
                      [  3.88000000e+02,   8.14867726e+12],
                      [  3.89000000e+02,   8.19941421e+12],
                      [  3.90000000e+02,   8.24993438e+12],
                      [  3.91000000e+02,   8.30023471e+12],
                      [  3.92000000e+02,   8.35031222e+12],
                      [  3.93000000e+02,   8.40016398e+12],
                      [  3.94000000e+02,   8.44978711e+12],
                      [  3.95000000e+02,   8.49917881e+12],
                      [  3.96000000e+02,   8.54833632e+12],
                      [  3.97000000e+02,   8.59725693e+12],
                      [  3.98000000e+02,   8.64593802e+12],
                      [  3.99000000e+02,   8.69437700e+12],
                      [  4.00000000e+02,   8.74257133e+12],
                      [  4.01000000e+02,   8.79051856e+12],
                      [  4.02000000e+02,   8.83821626e+12],
                      [  4.03000000e+02,   8.88566209e+12],
                      [  4.04000000e+02,   8.93285373e+12],
                      [  4.05000000e+02,   8.97978893e+12],
                      [  4.06000000e+02,   9.02646551e+12],
                      [  4.07000000e+02,   9.07288133e+12],
                      [  4.08000000e+02,   9.11903431e+12],
                      [  4.09000000e+02,   9.16492240e+12],
                      [  4.10000000e+02,   9.21054364e+12],
                      [  4.11000000e+02,   9.25589609e+12],
                      [  4.12000000e+02,   9.30097789e+12],
                      [  4.13000000e+02,   9.34578722e+12],
                      [  4.14000000e+02,   9.39032230e+12],
                      [  4.15000000e+02,   9.43458143e+12],
                      [  4.16000000e+02,   9.47856292e+12],
```

(continues on next page)

(continued from previous page)

```
                    [  4.17000000e+02,   9.52226517e+12],
                    [  4.18000000e+02,   9.56568661e+12],
                    [  4.19000000e+02,   9.60882571e+12],
                    [  4.20000000e+02,   9.65168102e+12],
                    [  4.21000000e+02,   9.69425111e+12],
                    [  4.22000000e+02,   9.73653461e+12],
                    [  4.23000000e+02,   9.77853020e+12],
                    [  4.24000000e+02,   9.82023659e+12],
                    [  4.25000000e+02,   9.86165257e+12],
                    [  4.26000000e+02,   9.90277693e+12],
                    [  4.27000000e+02,   9.94360856e+12],
                    [  4.28000000e+02,   9.98414634e+12],
                    [  4.29000000e+02,   1.00243892e+13],
                    [  4.30000000e+02,   1.00643363e+13],
                    [  4.31000000e+02,   1.01039864e+13],
                    [  4.32000000e+02,   1.01433388e+13],
                    [  4.33000000e+02,   1.01823926e+13],
                    [  4.34000000e+02,   1.02211468e+13],
                    [  4.35000000e+02,   1.02596009e+13],
                    [  4.36000000e+02,   1.02977539e+13],
                    [  4.37000000e+02,   1.03356052e+13],
                    [  4.38000000e+02,   1.03731541e+13],
                    [  4.39000000e+02,   1.04104000e+13],
                    [  4.40000000e+02,   1.04473423e+13],
                    [  4.41000000e+02,   1.04839805e+13],
                    [  4.42000000e+02,   1.05203140e+13],
                    [  4.43000000e+02,   1.05563424e+13],
                    [  4.44000000e+02,   1.05920652e+13],
                    [  4.45000000e+02,   1.06274821e+13],
                    [  4.46000000e+02,   1.06625927e+13],
                    [  4.47000000e+02,   1.06973967e+13],
                    [  4.48000000e+02,   1.07318937e+13],
                    [  4.49000000e+02,   1.07660835e+13],
                    [  4.50000000e+02,   1.07999660e+13],
                    [  4.51000000e+02,   1.08335408e+13],
                    [  4.52000000e+02,   1.08668080e+13],
                    [  4.53000000e+02,   1.08997673e+13],
                    [  4.54000000e+02,   1.09324187e+13],
                    [  4.55000000e+02,   1.09647621e+13],
                    [  4.56000000e+02,   1.09967975e+13],
                    [  4.57000000e+02,   1.10285249e+13],
                    [  4.58000000e+02,   1.10599443e+13],
                    [  4.59000000e+02,   1.10910559e+13],
                    [  4.60000000e+02,   1.11218598e+13],
                    [  4.61000000e+02,   1.11523560e+13],
                    [  4.62000000e+02,   1.11825447e+13],
                    [  4.63000000e+02,   1.12124262e+13],
                    [  4.64000000e+02,   1.12420006e+13],
                    [  4.65000000e+02,   1.12712681e+13],
                    [  4.66000000e+02,   1.13002292e+13],
                    [  4.67000000e+02,   1.13288840e+13],
                    [  4.68000000e+02,   1.13572329e+13],
                    [  4.69000000e+02,   1.13852762e+13],
                    [  4.70000000e+02,   1.14130144e+13],
                    [  4.71000000e+02,   1.14404478e+13],
                    [  4.72000000e+02,   1.14675768e+13],
                    [  4.73000000e+02,   1.14944020e+13],
                    [  4.74000000e+02,   1.15209237e+13],
                    [  4.75000000e+02,   1.15471425e+13],
                    [  4.76000000e+02,   1.15730590e+13],
                    [  4.77000000e+02,   1.15986736e+13],
```

(continues on next page)

(continued from previous page)

```
                [  4.78000000e+02,   1.16239869e+13],
                [  4.79000000e+02,   1.16489996e+13],
                [  4.80000000e+02,   1.16737122e+13],
                [  4.81000000e+02,   1.16981253e+13],
                [  4.82000000e+02,   1.17222397e+13],
                [  4.83000000e+02,   1.17460559e+13],
                [  4.84000000e+02,   1.17695747e+13],
                [  4.85000000e+02,   1.17927969e+13],
                [  4.86000000e+02,   1.18157230e+13],
                [  4.87000000e+02,   1.18383540e+13],
                [  4.88000000e+02,   1.18606904e+13],
                [  4.89000000e+02,   1.18827333e+13],
                [  4.90000000e+02,   1.19044832e+13],
                [  4.91000000e+02,   1.19259412e+13],
                [  4.92000000e+02,   1.19471079e+13],
                [  4.93000000e+02,   1.19679843e+13],
                [  4.94000000e+02,   1.19885712e+13],
                [  4.95000000e+02,   1.20088695e+13],
                [  4.96000000e+02,   1.20288802e+13],
                [  4.97000000e+02,   1.20486041e+13],
                [  4.98000000e+02,   1.20680421e+13],
                [  4.99000000e+02,   1.20871953e+13],
                [  5.00000000e+02,   1.21060645e+13],
                [  5.01000000e+02,   1.21246508e+13],
                [  5.02000000e+02,   1.21429552e+13],
                [  5.03000000e+02,   1.21609785e+13],
                [  5.04000000e+02,   1.21787220e+13],
                [  5.05000000e+02,   1.21961865e+13],
                [  5.06000000e+02,   1.22133731e+13],
                [  5.07000000e+02,   1.22302829e+13],
                [  5.08000000e+02,   1.22469170e+13],
                [  5.09000000e+02,   1.22632763e+13],
                [  5.10000000e+02,   1.22793620e+13],
                [  5.11000000e+02,   1.22951752e+13],
                [  5.12000000e+02,   1.23107171e+13],
                [  5.13000000e+02,   1.23259886e+13],
                [  5.14000000e+02,   1.23409909e+13],
                [  5.15000000e+02,   1.23557252e+13],
                [  5.16000000e+02,   1.23701926e+13],
                [  5.17000000e+02,   1.23843943e+13],
                [  5.18000000e+02,   1.23983314e+13],
                [  5.19000000e+02,   1.24120051e+13],
                [  5.20000000e+02,   1.24254166e+13],
                [  5.21000000e+02,   1.24385670e+13],
                [  5.22000000e+02,   1.24514576e+13],
                [  5.23000000e+02,   1.24640896e+13],
                [  5.24000000e+02,   1.24764641e+13],
                [  5.25000000e+02,   1.24885824e+13],
                [  5.26000000e+02,   1.25004457e+13],
                [  5.27000000e+02,   1.25120552e+13],
                [  5.28000000e+02,   1.25234122e+13],
                [  5.29000000e+02,   1.25345178e+13],
                [  5.30000000e+02,   1.25453735e+13],
                [  5.31000000e+02,   1.25559803e+13],
                [  5.32000000e+02,   1.25663396e+13],
                [  5.33000000e+02,   1.25764527e+13],
                [  5.34000000e+02,   1.25863207e+13],
                [  5.35000000e+02,   1.25959449e+13],
                [  5.36000000e+02,   1.26053268e+13],
                [  5.37000000e+02,   1.26144674e+13],
                [  5.38000000e+02,   1.26233681e+13],
```

(continues on next page)

(continued from previous page)

```
                    [  5.39000000e+02,   1.26320302e+13],
                    [  5.40000000e+02,   1.26404551e+13],
                    [  5.41000000e+02,   1.26486438e+13],
                    [  5.42000000e+02,   1.26565979e+13],
                    [  5.43000000e+02,   1.26643185e+13],
                    [  5.44000000e+02,   1.26718071e+13],
                    [  5.45000000e+02,   1.26790648e+13],
                    [  5.46000000e+02,   1.26860930e+13],
                    [  5.47000000e+02,   1.26928930e+13],
                    [  5.48000000e+02,   1.26994662e+13],
                    [  5.49000000e+02,   1.27058138e+13],
                    [  5.50000000e+02,   1.27119372e+13],
                    [  5.51000000e+02,   1.27178376e+13],
                    [  5.52000000e+02,   1.27235164e+13],
                    [  5.53000000e+02,   1.27289750e+13],
                    [  5.54000000e+02,   1.27342146e+13],
                    [  5.55000000e+02,   1.27392366e+13],
                    [  5.56000000e+02,   1.27440423e+13],
                    [  5.57000000e+02,   1.27486330e+13],
                    [  5.58000000e+02,   1.27530100e+13],
                    [  5.59000000e+02,   1.27571748e+13],
                    [  5.60000000e+02,   1.27611285e+13],
                    [  5.61000000e+02,   1.27648725e+13],
                    [  5.62000000e+02,   1.27684083e+13],
                    [  5.63000000e+02,   1.27717370e+13],
                    [  5.64000000e+02,   1.27748600e+13],
                    [  5.65000000e+02,   1.27777787e+13],
                    [  5.66000000e+02,   1.27804943e+13],
                    [  5.67000000e+02,   1.27830082e+13],
                    [  5.68000000e+02,   1.27853217e+13],
                    [  5.69000000e+02,   1.27874362e+13],
                    [  5.70000000e+02,   1.27893529e+13],
                    [  5.71000000e+02,   1.27910732e+13],
                    [  5.72000000e+02,   1.27925984e+13],
                    [  5.73000000e+02,   1.27939299e+13],
                    [  5.74000000e+02,   1.27950689e+13],
                    [  5.75000000e+02,   1.27960167e+13],
                    [  5.76000000e+02,   1.27967747e+13],
                    [  5.77000000e+02,   1.27973442e+13],
                    [  5.78000000e+02,   1.27977264e+13],
                    [  5.79000000e+02,   1.27979228e+13],
                    [  5.80000000e+02,   1.27979346e+13],
                    [  5.81000000e+02,   1.27977630e+13],
                    [  5.82000000e+02,   1.27974095e+13],
                    [  5.83000000e+02,   1.27968753e+13],
                    [  5.84000000e+02,   1.27961617e+13],
                    [  5.85000000e+02,   1.27952700e+13],
                    [  5.86000000e+02,   1.27942015e+13],
                    [  5.87000000e+02,   1.27929575e+13],
                    [  5.88000000e+02,   1.27915392e+13],
                    [  5.89000000e+02,   1.27899480e+13],
                    [  5.90000000e+02,   1.27881852e+13],
                    [  5.91000000e+02,   1.27862519e+13],
                    [  5.92000000e+02,   1.27841495e+13],
                    [  5.93000000e+02,   1.27818793e+13],
                    [  5.94000000e+02,   1.27794424e+13],
                    [  5.95000000e+02,   1.27768403e+13],
                    [  5.96000000e+02,   1.27740741e+13],
                    [  5.97000000e+02,   1.27711451e+13],
                    [  5.98000000e+02,   1.27680546e+13],
                    [  5.99000000e+02,   1.27648037e+13],
```

(continues on next page)

(continued from previous page)

```
                       [  6.00000000e+02,   1.27613938e+13],
                       [  6.01000000e+02,   1.27578261e+13],
                       [  6.02000000e+02,   1.27541018e+13],
                       [  6.03000000e+02,   1.27502222e+13],
                       [  6.04000000e+02,   1.27461885e+13],
                       [  6.05000000e+02,   1.27420020e+13],
                       [  6.06000000e+02,   1.27376637e+13],
                       [  6.07000000e+02,   1.27331750e+13],
                       [  6.08000000e+02,   1.27285371e+13],
                       [  6.09000000e+02,   1.27237512e+13],
                       [  6.10000000e+02,   1.27188185e+13],
                       [  6.11000000e+02,   1.27137402e+13],
                       [  6.12000000e+02,   1.27085175e+13],
                       [  6.13000000e+02,   1.27031516e+13],
                       [  6.14000000e+02,   1.26976436e+13],
                       [  6.15000000e+02,   1.26919949e+13],
                       [  6.16000000e+02,   1.26862064e+13],
                       [  6.17000000e+02,   1.26802795e+13],
                       [  6.18000000e+02,   1.26742153e+13],
                       [  6.19000000e+02,   1.26680149e+13],
                       [  6.20000000e+02,   1.26616795e+13],
                       [  6.21000000e+02,   1.26552103e+13],
                       [  6.22000000e+02,   1.26486085e+13],
                       [  6.23000000e+02,   1.26418751e+13],
                       [  6.24000000e+02,   1.26350113e+13],
                       [  6.25000000e+02,   1.26280183e+13],
                       [  6.26000000e+02,   1.26208972e+13],
                       [  6.27000000e+02,   1.26136491e+13],
                       [  6.28000000e+02,   1.26062751e+13],
                       [  6.29000000e+02,   1.25987764e+13],
                       [  6.30000000e+02,   1.25911540e+13],
                       [  6.31000000e+02,   1.25834092e+13],
                       [  6.32000000e+02,   1.25755429e+13],
                       [  6.33000000e+02,   1.25675563e+13],
                       [  6.34000000e+02,   1.25594505e+13],
                       [  6.35000000e+02,   1.25512265e+13],
                       [  6.36000000e+02,   1.25428855e+13],
                       [  6.37000000e+02,   1.25344285e+13],
                       [  6.38000000e+02,   1.25258566e+13],
                       [  6.39000000e+02,   1.25171709e+13],
                       [  6.40000000e+02,   1.25083724e+13],
                       [  6.41000000e+02,   1.24994622e+13],
                       [  6.42000000e+02,   1.24904413e+13],
                       [  6.43000000e+02,   1.24813108e+13],
                       [  6.44000000e+02,   1.24720718e+13],
                       [  6.45000000e+02,   1.24627252e+13],
                       [  6.46000000e+02,   1.24532721e+13],
                       [  6.47000000e+02,   1.24437136e+13],
                       [  6.48000000e+02,   1.24340506e+13],
                       [  6.49000000e+02,   1.24242842e+13],
                       [  6.50000000e+02,   1.24144153e+13],
                       [  6.51000000e+02,   1.24044450e+13],
                       [  6.52000000e+02,   1.23943743e+13],
                       [  6.53000000e+02,   1.23842042e+13],
                       [  6.54000000e+02,   1.23739356e+13],
                       [  6.55000000e+02,   1.23635696e+13],
                       [  6.56000000e+02,   1.23531072e+13],
                       [  6.57000000e+02,   1.23425492e+13],
                       [  6.58000000e+02,   1.23318967e+13],
                       [  6.59000000e+02,   1.23211506e+13],
                       [  6.60000000e+02,   1.23103120e+13],
```

(continues on next page)

(continued from previous page)

```
                    [  6.61000000e+02,   1.22993816e+13],
                    [  6.62000000e+02,   1.22883606e+13],
                    [  6.63000000e+02,   1.22772498e+13],
                    [  6.64000000e+02,   1.22660502e+13],
                    [  6.65000000e+02,   1.22547627e+13],
                    [  6.66000000e+02,   1.22433883e+13],
                    [  6.67000000e+02,   1.22319278e+13],
                    [  6.68000000e+02,   1.22203821e+13],
                    [  6.69000000e+02,   1.22087523e+13],
                    [  6.70000000e+02,   1.21970391e+13],
                    [  6.71000000e+02,   1.21852435e+13],
                    [  6.72000000e+02,   1.21733664e+13],
                    [  6.73000000e+02,   1.21614087e+13],
                    [  6.74000000e+02,   1.21493712e+13],
                    [  6.75000000e+02,   1.21372548e+13],
                    [  6.76000000e+02,   1.21250605e+13],
                    [  6.77000000e+02,   1.21127890e+13],
                    [  6.78000000e+02,   1.21004413e+13],
                    [  6.79000000e+02,   1.20880182e+13],
                    [  6.80000000e+02,   1.20755205e+13],
                    [  6.81000000e+02,   1.20629491e+13],
                    [  6.82000000e+02,   1.20503049e+13],
                    [  6.83000000e+02,   1.20375887e+13],
                    [  6.84000000e+02,   1.20248012e+13],
                    [  6.85000000e+02,   1.20119434e+13],
                    [  6.86000000e+02,   1.19990161e+13],
                    [  6.87000000e+02,   1.19860200e+13],
                    [  6.88000000e+02,   1.19729560e+13],
                    [  6.89000000e+02,   1.19598249e+13],
                    [  6.90000000e+02,   1.19466275e+13],
                    [  6.91000000e+02,   1.19333646e+13],
                    [  6.92000000e+02,   1.19200370e+13],
                    [  6.93000000e+02,   1.19066454e+13],
                    [  6.94000000e+02,   1.18931907e+13],
                    [  6.95000000e+02,   1.18796736e+13],
                    [  6.96000000e+02,   1.18660949e+13],
                    [  6.97000000e+02,   1.18524554e+13],
                    [  6.98000000e+02,   1.18387558e+13],
                    [  6.99000000e+02,   1.18249969e+13],
                    [  7.00000000e+02,   1.18111794e+13],
                    [  7.01000000e+02,   1.17973040e+13],
                    [  7.02000000e+02,   1.17833716e+13],
                    [  7.03000000e+02,   1.17693829e+13],
                    [  7.04000000e+02,   1.17553385e+13],
                    [  7.05000000e+02,   1.17412392e+13],
                    [  7.06000000e+02,   1.17270858e+13],
                    [  7.07000000e+02,   1.17128789e+13],
                    [  7.08000000e+02,   1.16986192e+13],
                    [  7.09000000e+02,   1.16843075e+13],
                    [  7.10000000e+02,   1.16699445e+13],
                    [  7.11000000e+02,   1.16555309e+13],
                    [  7.12000000e+02,   1.16410673e+13],
                    [  7.13000000e+02,   1.16265544e+13],
                    [  7.14000000e+02,   1.16119930e+13],
                    [  7.15000000e+02,   1.15973836e+13],
                    [  7.16000000e+02,   1.15827271e+13],
                    [  7.17000000e+02,   1.15680240e+13],
                    [  7.18000000e+02,   1.15532749e+13],
                    [  7.19000000e+02,   1.15384807e+13],
                    [  7.20000000e+02,   1.15236419e+13],
                    [  7.21000000e+02,   1.15087591e+13],
```

(continues on next page)

(continued from previous page)

```
                [  7.22000000e+02,   1.14938331e+13],
                [  7.23000000e+02,   1.14788644e+13],
                [  7.24000000e+02,   1.14638537e+13],
                [  7.25000000e+02,   1.14488017e+13],
                [  7.26000000e+02,   1.14337088e+13],
                [  7.27000000e+02,   1.14185759e+13],
                [  7.28000000e+02,   1.14034034e+13],
                [  7.29000000e+02,   1.13881921e+13],
                [  7.30000000e+02,   1.13729424e+13],
                [  7.31000000e+02,   1.13576551e+13],
                [  7.32000000e+02,   1.13423307e+13],
                [  7.33000000e+02,   1.13269698e+13],
                [  7.34000000e+02,   1.13115730e+13],
                [  7.35000000e+02,   1.12961409e+13],
                [  7.36000000e+02,   1.12806741e+13],
                [  7.37000000e+02,   1.12651731e+13],
                [  7.38000000e+02,   1.12496385e+13],
                [  7.39000000e+02,   1.12340710e+13],
                [  7.40000000e+02,   1.12184710e+13],
                [  7.41000000e+02,   1.12028391e+13],
                [  7.42000000e+02,   1.11871759e+13],
                [  7.43000000e+02,   1.11714819e+13],
                [  7.44000000e+02,   1.11557577e+13],
                [  7.45000000e+02,   1.11400038e+13],
                [  7.46000000e+02,   1.11242208e+13],
                [  7.47000000e+02,   1.11084092e+13],
                [  7.48000000e+02,   1.10925695e+13],
                [  7.49000000e+02,   1.10767023e+13],
                [  7.50000000e+02,   1.10608080e+13],
                [  7.51000000e+02,   1.10448872e+13],
                [  7.52000000e+02,   1.10289405e+13],
                [  7.53000000e+02,   1.10129683e+13],
                [  7.54000000e+02,   1.09969711e+13],
                [  7.55000000e+02,   1.09809495e+13],
                [  7.56000000e+02,   1.09649039e+13],
                [  7.57000000e+02,   1.09488348e+13],
                [  7.58000000e+02,   1.09327427e+13],
                [  7.59000000e+02,   1.09166282e+13],
                [  7.60000000e+02,   1.09004917e+13],
                [  7.61000000e+02,   1.08843336e+13],
                [  7.62000000e+02,   1.08681545e+13],
                [  7.63000000e+02,   1.08519548e+13],
                [  7.64000000e+02,   1.08357350e+13],
                [  7.65000000e+02,   1.08194956e+13],
                [  7.66000000e+02,   1.08032370e+13],
                [  7.67000000e+02,   1.07869596e+13],
                [  7.68000000e+02,   1.07706640e+13],
                [  7.69000000e+02,   1.07543506e+13],
                [  7.70000000e+02,   1.07380198e+13],
                [  7.71000000e+02,   1.07216721e+13],
                [  7.72000000e+02,   1.07053078e+13],
                [  7.73000000e+02,   1.06889276e+13],
                [  7.74000000e+02,   1.06725317e+13],
                [  7.75000000e+02,   1.06561206e+13],
                [  7.76000000e+02,   1.06396947e+13],
                [  7.77000000e+02,   1.06232545e+13],
                [  7.78000000e+02,   1.06068004e+13],
                [  7.79000000e+02,   1.05903327e+13],
                [  7.80000000e+02,   1.05738520e+13]],
               interpolator=SpragueInterpolator,
               interpolator_args={},
```

(continues on next page)

(continued from previous page)

```
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
```

colour.sd_constant

colour.**sd_constant**(*k*, *shape=SpectralShape(360, 780, 1)*, *dtype=<class 'numpy.float64'>*)

Returns a spectral distribution of given spectral shape filled with constant k values.

Parameters

- **k** (numeric) – Constant k to fill the spectral distribution with.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.
- **dtype** (type) – Data type used for the spectral distribution.

Returns Constant k to filled spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> sd = sd_constant(100)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
100.0
```

colour.sd_ones

colour.**sd_ones**(*shape=SpectralShape(360, 780, 1)*)

Returns a spectral distribution of given spectral shape filled with ones.

Parameters **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.

Returns Ones filled spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> sd = sd_ones()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
1.0
```

colour.sd_zeros

colour.**sd_zeros**(*shape=SpectralShape(360, 780, 1)*)

Returns a spectral distribution of given spectral shape filled with zeros.

Parameters **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.

Returns Zeros filled spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> sd = sd_zeros()
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[400]
0.0
```

colour.SD_GAUSSIAN_METHODS

colour.**SD_GAUSSIAN_METHODS = CaseInsensitiveMapping({'Normal': ..., 'FWHM': ...})**

Supported gaussian spectral distribution computation methods.

SD_GAUSSIAN_METHODS [CaseInsensitiveMapping] **{'Normal', 'FWHM'}**

colour.sd_gaussian

colour.**sd_gaussian**(*mu_peak_wavelength*, *sigma_fwhm*, *shape=SpectralShape(360, 780, 1)*, *method='Normal'*)

Returns a gaussian spectral distribution of given spectral shape using given method.

Parameters

- **mu_peak_wavelength** (numeric) – Mean wavelength μ the gaussian spectral distribution will peak at.
- **sigma_fwhm** (numeric) – Standard deviation $sigma$ of the gaussian spectral distribution or Full width at half maximum, i.e. width of the gaussian spectral distribution measured between those points on the y axis which are half the maximum amplitude.

- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.
- **method** (unicode, optional) – **{'Normal', 'FWHM'}**, Computation method.

Returns Gaussian spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> sd = sd_gaussian(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0000000...
>>> sd[530]
0.6065306...
>>> sd = sd_gaussian(555, 25, method='FWHM')
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
0.3678794...
```

colour.SD_SINGLE_LED_METHODS

colour.**SD_SINGLE_LED_METHODS = CaseInsensitiveMapping({'Ohno 2005': ...})**

Supported single *LED* spectral distribution computation methods.

SD_SINGLE_LED_METHODS [CaseInsensitiveMapping] **{'Ohno 2005'}**

colour.sd_single_led

colour.**sd_single_led**(*peak_wavelength*, *fwhm*, *shape=SpectralShape(360, 780, 1)*, *method='Ohno 2005'*)

Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to given method.

Parameters

- **peak_wavelength** (numeric) – Wavelength the single *LED* spectral distribution will peak at.
- **fwhm** (numeric) – Full width at half maximum, i.e. width of the underlying gaussian spectral distribution measured between those points on the *y* axis which are half the maximum amplitude.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.
- **method** (unicode, optional) – **{'Ohno 2005'}**, Computation method.

Returns Single *LED* spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_single_led(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0000000...
```

colour.SD_MULTI_LEDS_METHODS

colour.**SD_MULTI_LEDS_METHODS = CaseInsensitiveMapping({'Ohno 2005': ...})**

Supported multi *LED* spectral distribution computation methods.

SD_MULTI_LEDS_METHODS [CaseInsensitiveMapping] **{'Ohno 2005'}**

colour.sd_multi_leds

colour.**sd_multi_leds**(*peak_wavelengths*, *fwhm*, *peak_power_ratios=None*, *shape=SpectralShape(360, 780, 1)*, *method='Ohno 2005'*)

Returns a multi *LED* spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to given method.

Parameters

- **peak_wavelengths** (array_like) – Wavelengths the multi *LED* spectral distribution will peak at, i.e. the peaks for each generated single *LED* spectral distributions.
- **fwhm** (array_like) – Full widths at half maximum, i.e. widths of the underlying gaussian spectral distributions measured between those points on the *y* axis which are half the maximum amplitude.
- **peak_power_ratios** (array_like, optional) – Peak power ratios for each generated single *LED* spectral distributions.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.
- **method** (unicode, optional) – **{'Ohno 2005'}**, Computation method.

Returns Multi *LED* spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_multi_leds(
...     np.array([457, 530, 615]),
...     np.array([20, 30, 20]),
...     np.array([0.731, 1.000, 1.660]),
... )
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[500]
0.1295132...
```

`colour.colorimetry`

`blackbody_spectral_radiance`(wavelength, ...)	Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n.
`daylight_locus_function`(x_D)	Returns the daylight locus as *CIE xy* chromaticity coordinates.
`sd_gaussian_normal`(mu, sigma[, shape])	Returns a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation $sigma$.
`sd_gaussian_fwhm`(peak_wavelength, fwhm[, shape])	Returns a gaussian spectral distribution of given spectral shape at given peak wavelength and full width at half maximum.
`sd_single_led_Ohno2005`(peak_wavelength, fwhm)	Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to *Ohno (2005)* method.
`sd_multi_leds_Ohno2005`(peak_wavelengths, fwhm)	Returns a multi *LED* spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to *Ohno (2005)* method.
`sds_and_multi_sds_to_sds`(sds)	Converts given spectral and multi-spectral distributions to a flat list of spectral distributions.
`sds_and_multi_sds_to_multi_sds`(sds)	Converts given spectral and multi-spectral distributions to multi-spectral distributions.

colour.colorimetry.blackbody_spectral_radiance

colour.colorimetry.**blackbody_spectral_radiance**(*wavelength*, *temperature*, *c1=3.741771e-16*, *c2=0.014388*, *n=1*)

Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n.

Parameters

- **wavelength** (numeric or array_like) – Wavelength in meters.
- **temperature** (numeric or array_like) – Temperature $T[K]$ in kelvin degrees.
- **c1** (numeric or array_like, optional) – The official value of $c1$ is provided by the Committee on Data for Science and Technology (CODATA) and is $c1 = 3,741771x10.16\ W/m_2$ *(Mohr and Taylor, 2000)*.
- **c2** (numeric or array_like, optional) – Since T is measured on the International Temperature Scale, the value of $c2$ used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et aI., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **n** (numeric or array_like, optional) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Radiance in *watts per steradian per square metre*.

Return type numeric or ndarray

Notes

- The following form implementation is expressed in term of wavelength.
- The SI unit of radiance is *watts per steradian per square metre*.

References

[CIET14804c]

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
```

colour.colorimetry.daylight_locus_function

colour.colorimetry.**daylight_locus_function**(*x_D*)

Returns the daylight locus as *CIE xy* chromaticity coordinates.

Parameters **x_D** (numeric or array_like) – *x* chromaticity coordinates

Returns Daylight locus as *CIE xy* chromaticity coordinates.

Return type numeric or array_like

References

[WS00a]

Examples

```
>>> daylight_locus_function(0.31270)
0.3291051...
```

colour.colorimetry.sd_gaussian_normal

colour.colorimetry.**sd_gaussian_normal**(*mu*, *sigma*, *shape=SpectralShape(360, 780, 1)*)

Returns a gaussian spectral distribution of given spectral shape at given mean wavelength μ and standard deviation $sigma$.

Parameters

- **mu** (numeric) – Mean wavelength μ the gaussian spectral distribution will peak at.
- **sigma** (numeric) – Standard deviation $sigma$ of the gaussian spectral distribution.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.

Returns Gaussian spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> sd = sd_gaussian_normal(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0000000...
>>> sd[530]
0.6065306...
```

colour.colorimetry.sd_gaussian_fwhm

colour.colorimetry.**sd_gaussian_fwhm**(*peak_wavelength*, *fwhm*, *shape=SpectralShape(360, 780, 1)*)

Returns a gaussian spectral distribution of given spectral shape at given peak wavelength and full width at half maximum.

Parameters

- **peak_wavelength** (numeric) – Wavelength the gaussian spectral distribution will peak at.

- **fwhm** (numeric) – Full width at half maximum, i.e. width of the gaussian spectral distribution measured between those points on the *y* axis which are half the maximum amplitude.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.

Returns Gaussian spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by colour.DEFAULT_SPECTRAL_SHAPE attribute.

Examples

```
>>> sd = sd_gaussian_fwhm(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0
>>> sd[530]
0.3678794...
```

colour.colorimetry.sd_single_led_Ohno2005

colour.colorimetry.**sd_single_led_Ohno2005**(*peak_wavelength*, *fwhm*, *shape=SpectralShape(360, 780, 1)*)

Returns a single *LED* spectral distribution of given spectral shape at given peak wavelength and full width at half maximum according to *Ohno (2005)* method.

Parameters

- **peak_wavelength** (numeric) – Wavelength the single *LED* spectral distribution will peak at.
- **fwhm** (numeric) – Full width at half maximum, i.e. width of the underlying gaussian spectral distribution measured between those points on the *y* axis which are half the maximum amplitude.
- **shape** (SpectralShape, optional) – Spectral shape used to create the spectral distribution.

Returns Single *LED* spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_single_led_Ohno2005(555, 25)
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[555]
1.0000000...
```

colour.colorimetry.sd_multi_leds_Ohno2005

colour.colorimetry.**sd_multi_leds_Ohno2005**(*peak_wavelengths*, *fwhm*, *peak_power_ratios=None*, *shape=SpectralShape(360, 780, 1)*)

Returns a multi *LED* spectral distribution of given spectral shape at given peak wavelengths and full widths at half maximum according to *Ohno (2005)* method.

The multi *LED* spectral distribution is generated using many single *LED* spectral distributions generated with `colour.sd_single_led_Ohno2005()` definition.

Parameters

- **peak_wavelengths** (`array_like`) – Wavelengths the multi *LED* spectral distribution will peak at, i.e. the peaks for each generated single *LED* spectral distributions.
- **fwhm** (`array_like`) – Full widths at half maximum, i.e. widths of the underlying gaussian spectral distributions measured between those points on the *y* axis which are half the maximum amplitude.
- **peak_power_ratios** (`array_like, optional`) – Peak power ratios for each generated single *LED* spectral distributions.
- **shape** (`SpectralShape, optional`) – Spectral shape used to create the spectral distribution.

Returns Multi *LED* spectral distribution.

Return type *SpectralDistribution*

Notes

- By default, the spectral distribution will use the shape given by `colour.DEFAULT_SPECTRAL_SHAPE` attribute.

References

[Ohn05], [OD08]

Examples

```
>>> sd = sd_multi_leds_Ohno2005(
...     np.array([457, 530, 615]),
...     np.array([20, 30, 20]),
...     np.array([0.731, 1.000, 1.660]),
... )
>>> sd.shape
SpectralShape(360.0, 780.0, 1.0)
>>> sd[500]
0.1295132...
```

colour.colorimetry.sds_and_multi_sds_to_sds

colour.colorimetry.**sds_and_multi_sds_to_sds**(*sds*)

Converts given spectral and multi-spectral distributions to a flat list of spectral distributions.

Parameters **sds** (array_like) – Spectral and multi-spectral distributions to convert to a flat list of spectral distributions.

Returns Flat list of spectral distributions.

Return type list

Examples

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd_1 = SpectralDistribution(data)
>>> sd_2 = SpectralDistribution(data)
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> multi_sds_1 = MultiSpectralDistributions(data)
>>> multi_sds_2 = MultiSpectralDistributions(data)
>>> len(sds_and_multi_sds_to_sds([sd_1, sd_2, multi_sds_1, multi_sds_2]))
8
```

colour.colorimetry.sds_and_multi_sds_to_multi_sds

colour.colorimetry.**sds_and_multi_sds_to_multi_sds**(*sds*)

Converts given spectral and multi-spectral distributions to multi-spectral distributions.

The spectral and multi-spectral distributions will be aligned to the intersection of their spectral shapes.

Parameters **sds** (array_like) – Spectral and multi-spectral distributions to convert to multi-spectral distributions.

Returns Multi-spectral distributions.

Return type *MultiSpectralDistributions*

Examples

```
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd_1 = SpectralDistribution(data)
>>> sd_2 = SpectralDistribution(data)
>>> data = {
...     500: (0.004900, 0.323000, 0.272000),
...     510: (0.009300, 0.503000, 0.158200),
...     520: (0.063270, 0.710000, 0.078250),
...     530: (0.165500, 0.862000, 0.042160),
...     540: (0.290400, 0.954000, 0.020300),
...     550: (0.433450, 0.994950, 0.008750),
...     560: (0.594500, 0.995000, 0.003900)
... }
>>> multi_sds_1 = MultiSpectralDistributions(data)
>>> multi_sds_2 = MultiSpectralDistributions(data)
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True, linewidth=160):
...     sds_and_multi_sds_to_multi_sds(
...         [sd_1, sd_2, multi_sds_1, multi_sds_2])
MultiSpectralDistributions([[ 500.        ,    0.0651    ...,    0.0651    ...,    0.0049    ...,␣
↪   0.323     ...,    0.272     ...,    0.0049    ...,    0.323     ...,    0.272     ...],
                           [ 510.        ,    0.0676692...,    0.0676692...,    0.0093    ...,␣
↪   0.503     ...,    0.1582    ...,    0.0093    ...,    0.503     ...,    0.1582    ...],
                           [ 520.        ,    0.0705    ...,    0.0705    ...,    0.06327   ...,␣
↪   0.71      ...,    0.07825   ...,    0.06327   ...,    0.71      ...,    0.07825   ...],
                           [ 530.        ,    0.0737808...,    0.0737808...,    0.1655    ...,␣
↪   0.862     ...,    0.04216   ...,    0.1655    ...,    0.862     ...,    0.04216   ...],
                           [ 540.        ,    0.0772    ...,    0.0772    ...,    0.2904    ...,␣
↪   0.954     ...,    0.0203    ...,    0.2904    ...,    0.954     ...,    0.0203    ...],
                           [ 550.        ,    0.0806671...,    0.0806671...,    0.43345   ...,␣
↪   0.99495   ...,    0.00875   ...,    0.43345   ...,    0.99495   ...,    0.00875   ...],
                           [ 560.        ,    0.087     ...,    0.087     ...,    0.5945    ...,␣
↪   0.995     ...,    0.0039    ...,    0.5945    ...,    0.995     ...,    0.0039    ...]],
                          labels=['SpectralDistribution (...)', 'SpectralDistribution (...)',
↪'0 - SpectralDistribution (...)', '1 - SpectralDistribution (...)', '2 -␣
↪SpectralDistribution (...)', '0 - SpectralDistribution (...)', '1 - SpectralDistribution (...
↪)', '2 - SpectralDistribution (...)'],
                          interpolator=SpragueInterpolator,
                          interpolator_args={},
```

(continues on next page)

(continued from previous page)

```
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
```

Aliases

`colour.colorimetry`

`planck_law`(wavelength, temperature[, c1, c2, n])	Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n.

colour.colorimetry.planck_law

`colour.colorimetry.`**`planck_law`**(*wavelength*, *temperature*, *c1=3.741771e-16*, *c2=0.014388*, *n=1*)

Returns the spectral radiance of a blackbody at thermodynamic temperature $T[K]$ in a medium having index of refraction n.

Parameters

- **wavelength** (`numeric or array_like`) – Wavelength in meters.
- **temperature** (`numeric or array_like`) – Temperature $T[K]$ in kelvin degrees.
- **c1** (`numeric or array_like, optional`) – The official value of $c1$ is provided by the Committee on Data for Science and Technology (CODATA) and is $c1 = 3,741771x10.16\ W/m_2$ *(Mohr and Taylor, 2000)*.
- **c2** (`numeric or array_like, optional`) – Since T is measured on the International Temperature Scale, the value of $c2$ used in colorimetry should follow that adopted in the current International Temperature Scale (ITS-90) *(Preston-Thomas, 1990; Mielenz et al., 1991)*, namely $c2 = 1,4388x10.2\ m/K$.
- **n** (`numeric or array_like, optional`) – Medium index of refraction. For dry air at 15C and 101 325 Pa, containing 0,03 percent by volume of carbon dioxide, it is approximately 1,00028 throughout the visible region although *CIE 15:2004* recommends using $n = 1$.

Returns Radiance in *watts per steradian per square metre*.

Return type numeric or ndarray

Notes

- The following form implementation is expressed in term of wavelength.
- The SI unit of radiance is *watts per steradian per square metre*.

References

[CIET14804c]

Examples

```
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> planck_law(500 * 1e-9, 5500)
20472701909806.5...
```

Conversion to Tristimulus Values

colour

sd_to_XYZ(sd[, cmfs, illuminant, k, method])	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.
SD_TO_XYZ_METHODS	Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.
multi_sds_to_XYZ(msds[, cmfs, illuminant, ...])	Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.
MULTI_SD_TO_XYZ_METHODS	Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.
wavelength_to_XYZ(wavelength[, cmfs])	Converts given wavelength λ to *CIE XYZ* tristimulus values using given colour matching functions.

colour.sd_to_XYZ

colour.**sd_to_XYZ**(*sd*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*, *method='ASTM E308'*, ***kwargs*)

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions, illuminant and method.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **method** (unicode, optional) – **{'ASTM E308', 'Integration'}**, Computation method.

Other Parameters

- **mi_5nm_omission_method** (*bool, optional*) – {colour.colorimetry.sd_to_XYZ_ASTME308()}, 5 nm measurement intervals spectral distribution conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (*bool, optional*) – {colour.colorimetry.sd_to_XYZ_ASTME308()}, 20 nm measurement intervals spectral distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **use_practice_range** (*bool, optional*) – {colour.colorimetry.sd_to_XYZ_ASTME308()}, Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational11], [ASTMInternational15], [WS00f]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ(sd, cmfs, illuminant)
...
array([ 10.8399031...,   9.6840375...,   6.2164159...])
>>> sd_to_XYZ(sd, cmfs, illuminant, use_practice_range=False)
...
```

(continues on next page)

(continued from previous page)

```
array([ 10.8399852...,   9.6840602...,   6.2164085...])
>>> sd_to_XYZ(sd, cmfs, illuminant, method='Integration')
...
array([ 10.8401846...,   9.6837311...,   6.2120912...])
```

colour.SD_TO_XYZ_METHODS

colour.**SD_TO_XYZ_METHODS = CaseInsensitiveMapping({'ASTM E308': ..., 'Integration': ..., 'astm2015': ...**

Supported spectral distribution to *CIE XYZ* tristimulus values conversion methods.

References

[ASTMInternational11], [ASTMInternational15], [WS00f]

SD_TO_XYZ_METHODS [CaseInsensitiveMapping] **{'ASTM E308', 'Integration'}**

Aliases:

- 'astm2015': 'ASTM E308'

colour.multi_sds_to_XYZ

colour.**multi_sds_to_XYZ**(*msds*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*, *method='ASTM E308'*, ***kwargs*)

Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant. For the *Integration* method, the multi-spectral distributions can be either a `colour.MultiSpectralDistributions` class instance or an *array_like* in which case the `shape` must be passed.

Parameters

- **msds** (`MultiSpectralDistributions or array_like`) – Multi-spectral distributions, if an *array_like* the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, `msds` shape should be (384, 512, 77).
- **cmfs** (`XYZ_ColourMatchingFunctions`) – Standard observer colour matching functions.
- **illuminant** (`SpectralDistribution, optional`) – Illuminant spectral distribution.
- **k** (`numeric, optional`) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **method** (`unicode, optional`) – **{'ASTM E308', 'Integration'}**, Computation method.

Other Parameters

- **use_practice_range** (*bool, optional*) – {colour.colorimetry.multi_sds_to_XYZ_ASTME308()}, Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.
- **mi_5nm_omission_method** (*bool, optional*) – {colour.colorimetry.multi_sds_to_XYZ_ASTME308()}, 5 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (*bool, optional*) – {colour.colorimetry.multi_sds_to_XYZ_ASTME308()}, 20 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **shape** (*SpectralShape, optional*) – {colour.colorimetry.multi_sds_to_XYZ_integration()}, Spectral shape of the multi-spectral distributions array $msds$, `cmfs` and `illuminant` will be aligned to it.

Returns *CIE XYZ* tristimulus values, for a 512x384 multi-spectral image with 77 wavelengths, the output shape will be (384, 512, 3).

Return type array_like

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- The code path using the *array_like* multi-spectral distributions produces results different to the code path using a colour.MultiSpectralDistributions class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.

References

[ASTMInternational11], [ASTMInternational15], [WS00f]

Examples

```
>>> shape = SpectralShape(400, 700, 60)
>>> data = np.array([
...     [0.0137, 0.0159, 0.0096, 0.0111, 0.0179, 0.1057, 0.0433,
...      0.0258, 0.0248, 0.0186, 0.0310, 0.0473],
...     [0.0913, 0.3145, 0.2582, 0.0709, 0.2971, 0.4620, 0.2683,
...      0.0831, 0.1203, 0.1292, 0.1682, 0.3221],
...     [0.0152, 0.0842, 0.4139, 0.0220, 0.5630, 0.1918, 0.2373,
...      0.0430, 0.0054, 0.0079, 0.3719, 0.2268],
...     [0.0281, 0.0907, 0.2228, 0.1249, 0.2375, 0.5625, 0.0518,
...      0.3230, 0.0065, 0.4006, 0.0861, 0.3161],
...     [0.1918, 0.7103, 0.0041, 0.1817, 0.0024, 0.4209, 0.0118,
...      0.2302, 0.1860, 0.9404, 0.0041, 0.1124],
...     [0.0430, 0.0437, 0.3744, 0.0020, 0.5819, 0.0027, 0.0823,
...      0.0081, 0.3625, 0.3213, 0.7849, 0.0024],
... ])
>>> msds = MultiSpectralDistributions(data, shape.range())
>>> multi_sds_to_XYZ(msds, method='Integration', shape=shape)
```

(continues on next page)

(continued from previous page)

```
...
array([[  8.2415862...,   4.2543993...,   7.6100842...],
       [ 29.6144619...,  16.1158465...,  25.9015472...],
       [ 16.6799560...,  27.2350547...,  22.9413337...],
       [ 12.5597688...,   9.0667136...,   5.9670327...],
       [ 18.5804689...,  33.6618109...,  26.9249733...],
       [ 47.7113308...,  40.4573249...,  39.6439145...],
       [  7.830207 ...,  12.3689624...,  23.3742655...],
       [ 24.1695370...,  20.0629815...,   7.2718670...],
       [  7.2333751...,   2.7982097...,  10.0688374...],
       [ 48.7358074...,  30.2417164...,  10.6753233...],
       [  8.3231013...,  18.6791507...,  15.8228184...],
       [ 24.6452277...,  26.0809382...,  27.7106399...]])
>>> msds = np.array([
...     [
...         [0.0137, 0.0913, 0.0152, 0.0281, 0.1918, 0.0430],
...         [0.0159, 0.3145, 0.0842, 0.0907, 0.7103, 0.0437],
...         [0.0096, 0.2582, 0.4139, 0.2228, 0.0041, 0.3744],
...         [0.0111, 0.0709, 0.0220, 0.1249, 0.1817, 0.0020],
...         [0.0179, 0.2971, 0.5630, 0.2375, 0.0024, 0.5819],
...         [0.1057, 0.4620, 0.1918, 0.5625, 0.4209, 0.0027],
...     ],
...     [
...         [0.0433, 0.2683, 0.2373, 0.0518, 0.0118, 0.0823],
...         [0.0258, 0.0831, 0.0430, 0.3230, 0.2302, 0.0081],
...         [0.0248, 0.1203, 0.0054, 0.0065, 0.1860, 0.3625],
...         [0.0186, 0.1292, 0.0079, 0.4006, 0.9404, 0.3213],
...         [0.0310, 0.1682, 0.3719, 0.0861, 0.0041, 0.7849],
...         [0.0473, 0.3221, 0.2268, 0.3161, 0.1124, 0.0024],
...     ],
... ])
>>> multi_sds_to_XYZ(msds, method='Integration', shape=shape)
...
array([[[  7.6862675...,   4.0925470...,   8.4950412...],
        [ 27.4119366...,  15.5014764...,  29.2825122...],
        [ 17.1283666...,  27.7798651...,  25.5232032...],
        [ 11.9824544...,   8.8127109...,   6.6518695...],
        [ 19.1030682...,  34.4597818...,  29.7653804...],
        [ 46.8243374...,  39.9551652...,  43.6541858...]],

       [[  8.0978189...,  12.7544378...,  25.8004512...],
        [ 23.4360673...,  19.6127966...,   7.9342408...],
        [  7.0933208...,   2.7894394...,  11.1527704...],
        [ 45.6313772...,  29.0068105...,  11.9934522...],
        [  8.9327884...,  19.4008147...,  17.1534186...],
        [ 24.6610235...,  26.1093760...,  30.7298791...]]])
```

colour.MULTI_SD_TO_XYZ_METHODS

colour.**MULTI_SD_TO_XYZ_METHODS = CaseInsensitiveMapping({'ASTM E308': ..., 'Integration': ..., 'astm2015**

Supported multi-spectral array to *CIE XYZ* tristimulus values conversion methods.

References

[ASTMInternational11], [ASTMInternational15], [WS00f]

MULTI_SD_TO_XYZ_METHODS [CaseInsensitiveMapping] **{'ASTM E308', 'Integration'}**

Aliases:

- 'astm2015': 'ASTM E308'

colour.wavelength_to_XYZ

colour.**wavelength_to_XYZ**(*wavelength*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Converts given wavelength λ to *CIE XYZ* tristimulus values using given colour matching functions.

If the wavelength λ is not available in the colour matching function, its value will be calculated according to *CIE 15:2004* recommendation: the method developed by *Sprague (1880)* will be used for interpolating functions having a uniformly spaced independent variable and the *Cubic Spline* method for non-uniformly spaced independent variable.

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in nm.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Raises ValueError – If wavelength λ is not contained in the colour matching functions domain.

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> from colour import CMFS
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> wavelength_to_XYZ(480, cmfs)
array([ 0.09564  ,  0.13902  ,  0.8129501...])
>>> wavelength_to_XYZ(480.5, cmfs)
array([ 0.0914287...,  0.1418350...,  0.7915726...])
```

ASTM E308-15

colour.colorimetry

sd_to_XYZ_ASTME308(sd[, cmfs, illuminant, ...])	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.
multi_sds_to_XYZ_ASTME308(msds[, cmfs, ...])	Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

colour.colorimetry.sd_to_XYZ_ASTME308

colour.colorimetry.**sd_to_XYZ_ASTME308**(*sd*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *use_practice_range=True*, *mi_5nm_omission_method=True*, *mi_20nm_interpolation_method=True*, *k=None*)

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **use_practice_range** (bool, optional) – Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.
- **mi_5nm_omission_method** (bool, optional) – 5 nm measurement intervals spectral distribution conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (bool, optional) – 20 nm measurement intervals spectral distribution conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

Warning:

- The tables of tristimulus weighting factors are cached in `colour.colorimetry.tristimulus._TRISTIMULUS_WEIGHTING_FACTORS_CACHE` attribute. Their identifier key is defined by the colour matching functions and illuminant names along with the current shape and normalisation constant k such as: *CIE 1964 10 Degree Standard Observer, A, (360.0, 830.0, 10.0), None* Considering the above, one should be mindful that using similar colour matching functions and illuminant names but with different spectral data will lead to unexpected behaviour.

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational15]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ_ASTME308(sd, cmfs, illuminant)
...
array([ 10.8399031...,   9.6840375...,   6.2164159...])
```

colour.colorimetry.multi_sds_to_XYZ_ASTME308

colour.colorimetry.**multi_sds_to_XYZ_ASTME308**(*msds*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *use_practice_range=True*, *mi_5nm_omission_method=True*, *mi_20nm_interpolation_method=True*, *k=None*)

Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to practise *ASTM E308-15* method.

Parameters

- **msds** (MultiSpectralDistributions or array_like) – Multi-spectral distributions.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **use_practice_range** (bool, optional) – Practise *ASTM E308-15* working wavelengths range is [360, 780], if *True* this argument will trim the colour matching functions appropriately.
- **mi_5nm_omission_method** (bool, optional) – 5 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a 5 nm version of the colour matching functions instead of a table of tristimulus weighting factors.
- **mi_20nm_interpolation_method** (bool, optional) – 20 nm measurement intervals multi-spectral distributions conversion to tristimulus values will use a dedicated interpolation method instead of a table of tristimulus weighting factors.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **shape** (SpectralShape, optional) – Spectral shape of the multi-spectral distributions, cmfs and illuminant will be aligned to it.

Returns *CIE XYZ* tristimulus values.

Return type array_like

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- The code path using the *array_like* multi-spectral distributions produces results different to the code path using a `colour.MultiSpectralDistributions` class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.

References

[WS00f]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> shape = SpectralShape(400, 700, 60)
>>> D65 = ILLUMINANTS_SDS['D65']
>>> data = np.array([
...     [0.0137, 0.0159, 0.0096, 0.0111, 0.0179, 0.1057, 0.0433,
...      0.0258, 0.0248, 0.0186, 0.0310, 0.0473],
...     [0.0913, 0.3145, 0.2582, 0.0709, 0.2971, 0.4620, 0.2683,
...      0.0831, 0.1203, 0.1292, 0.1682, 0.3221],
...     [0.0152, 0.0842, 0.4139, 0.0220, 0.5630, 0.1918, 0.2373,
...      0.0430, 0.0054, 0.0079, 0.3719, 0.2268],
...     [0.0281, 0.0907, 0.2228, 0.1249, 0.2375, 0.5625, 0.0518,
...      0.3230, 0.0065, 0.4006, 0.0861, 0.3161],
...     [0.1918, 0.7103, 0.0041, 0.1817, 0.0024, 0.4209, 0.0118,
...      0.2302, 0.1860, 0.9404, 0.0041, 0.1124],
...     [0.0430, 0.0437, 0.3744, 0.0020, 0.5819, 0.0027, 0.0823,
...      0.0081, 0.3625, 0.3213, 0.7849, 0.0024],
... ])
>>> msds = MultiSpectralDistributions(data, shape.range())
>>> msds = msds.align(SpectralShape(400, 700, 20))
>>> multi_sds_to_XYZ_ASTME308(msds, illuminant=D65)
...
array([[  7.5054003...,   3.9556401...,   8.3911571...],
       [ 26.9413788...,  15.0983954...,  28.6698509...],
       [ 16.7055672...,  28.2093633...,  25.6615580...],
       [ 11.570967 ...,   8.6443158...,   6.5602307...],
       [ 18.7440223...,  35.0632090...,  30.1846483...],
       [ 45.1222564...,  39.6234504...,  43.5905694...],
       [  8.1797385...,  13.0953249...,  25.9383064...],
       [ 22.4455034...,  19.311139 ...,   7.9320266...],
       [  6.5766847...,   2.5305175...,  11.0749224...],
       [ 43.9100652...,  27.9994198...,  11.6878495...],
       [  8.5504477...,  19.6918148...,  17.7437346...],
       [ 23.8870261...,  26.2147602...,  30.6365272...]])
```

Ancillary Objects

`colour.colorimetry`

sd_to_XYZ_tristimulus_weighting_factors_ASTME308(sd, ...)	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant using a table of tristimulus weighting factors according to practise *ASTM E308-15* method.
adjust_tristimulus_weighting_factors_ASTME308(W, ...)	Adjusts given table of tristimulus weighting factors to account for a shorter wavelengths range of the test spectral shape compared to the reference spectral shape using practise *ASTM E308-15* method: Weights at the wavelengths for which data are not available are added to the weights at the shortest and longest wavelength for which spectral data are available.
lagrange_coefficients_ASTME2022([interval, ...])	Computes the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.
tristimulus_weighting_factors_ASTME2022(...)	Returns a table of tristimulus weighting factors for given colour matching functions and illuminant using practise *ASTM E2022-11* method.

colour.colorimetry.sd_to_XYZ_tristimulus_weighting_factors_ASTME308

colour.colorimetry.**sd_to_XYZ_tristimulus_weighting_factors_ASTME308**(*sd*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*)

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant using a table of tristimulus weighting factors according to practise *ASTM E308-15* method.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[ASTMInternational15]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ_tristimulus_weighting_factors_ASTME308(
...     sd, cmfs, illuminant)
array([ 10.8402899...,   9.6843539...,   6.2160858...])
```

colour.colorimetry.adjust_tristimulus_weighting_factors_ASTME308

colour.colorimetry.**adjust_tristimulus_weighting_factors_ASTME308**(*W*, *shape_r*, *shape_t*)

Adjusts given table of tristimulus weighting factors to account for a shorter wavelengths range of the test spectral shape compared to the reference spectral shape using practise *ASTM E308-15* method: Weights at the wavelengths for which data are not available are added to the weights at the shortest and longest wavelength for which spectral data are available.

Parameters

- **W** (array_like) – Tristimulus weighting factors table.
- **shape_r** (SpectralShape) – Reference spectral shape.
- **shape_t** (SpectralShape) – Test spectral shape.

Returns Adjusted tristimulus weighting factors.

Return type ndarray

References

[ASTMInternational15]

Examples

```
>>> from colour import (CMFS, sd_CIE_standard_illuminant_A,
...     SpectralDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = CMFS['CIE 1964 10 Degree Standard Observer']
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> W = tristimulus_weighting_factors_ASTME2022(
...     cmfs, A, SpectralShape(360, 830, 20))
>>> with numpy_print_options(suppress=True):
...     adjust_tristimulus_weighting_factors_ASTME308(
...         W,  SpectralShape(360, 830, 20), SpectralShape(400, 700, 20))
...
array([[  0.0509543...,   0.0040971...,   0.2144280...],
       [  0.7734225...,   0.0779839...,   3.6965732...],
       [  1.9000905...,   0.3037005...,   9.7554195...],
       [  1.9707727...,   0.8552809...,  11.4867325...],
       [  0.7183623...,   2.1457000...,   6.7845806...],
       [  0.0426667...,   4.8985328...,   2.3208000...],
       [  1.5223302...,   9.6471138...,   0.7430671...],
       [  5.6770329...,  14.4609708...,   0.1958194...],
       [ 12.4451744...,  17.4742541...,   0.0051827...],
       [ 20.5535772...,  17.5838219...,  -0.0026512...],
       [ 25.3315384...,  14.8957035...,   0.        ...],
       [ 21.5711570...,  10.0796619...,   0.        ...],
       [ 12.1785817...,   5.0680655...,   0.        ...],
       [  4.6675746...,   1.8303239...,   0.        ...],
       [  1.3236117...,   0.5129694...,   0.        ...],
       [  0.4171109...,   0.1618194...,   0.        ...]])
```

colour.colorimetry.lagrange_coefficients_ASTME2022

colour.colorimetry.**lagrange_coefficients_ASTME2022**(*interval=10*, *interval_type='inner'*)

Computes the *Lagrange Coefficients* for given interval size using practise *ASTM E2022-11* method.

Parameters

- **interval** (int) – Interval size in nm.
- **interval_type** (unicode, optional) – **{'inner', 'boundary'}**, If the interval is an *inner* interval *Lagrange Coefficients* are computed for degree 4. Degree 3 is used for a *boundary* interval.

Returns *Lagrange Coefficients*.

Return type ndarray

References

[ASTMInternational11]

Examples

```
>>> lagrange_coefficients_ASTME2022(10, 'inner')
...
array([[-0.028...,  0.940...,  0.104..., -0.016...],
       [-0.048...,  0.864...,  0.216..., -0.032...],
       [-0.059...,  0.773...,  0.331..., -0.045...],
       [-0.064...,  0.672...,  0.448..., -0.056...],
       [-0.062...,  0.562...,  0.562..., -0.062...],
       [-0.056...,  0.448...,  0.672..., -0.064...],
       [-0.045...,  0.331...,  0.773..., -0.059...],
       [-0.032...,  0.216...,  0.864..., -0.048...],
       [-0.016...,  0.104...,  0.940..., -0.028...]])
>>> lagrange_coefficients_ASTME2022(10, 'boundary')
...
array([[ 0.85...,  0.19..., -0.04...],
       [ 0.72...,  0.36..., -0.08...],
       [ 0.59...,  0.51..., -0.10...],
       [ 0.48...,  0.64..., -0.12...],
       [ 0.37...,  0.75..., -0.12...],
       [ 0.28...,  0.84..., -0.12...],
       [ 0.19...,  0.91..., -0.10...],
       [ 0.12...,  0.96..., -0.08...],
       [ 0.05...,  0.99..., -0.04...]])
```

colour.colorimetry.tristimulus_weighting_factors_ASTME2022

colour.colorimetry.**tristimulus_weighting_factors_ASTME2022**(*cmfs*, *illuminant*, *shape*, *k=None*)

Returns a table of tristimulus weighting factors for given colour matching functions and illuminant using practise *ASTM E2022-11* method.

The computed table of tristimulus weighting factors should be used with spectral data that has been corrected for spectral bandpass dependence.

Parameters

- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution) – Illuminant spectral distribution.
- **shape** (SpectralShape) – Shape used to build the table, only the interval is needed.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns Tristimulus weighting factors table.

Return type ndarray

Raises `ValueError` – If the colour matching functions or illuminant intervals are not equal to 1 nm.

Warning:

- The tables of tristimulus weighting factors are cached in `colour.colorimetry.tristimulus._TRISTIMULUS_WEIGHTING_FACTORS_CACHE` attribute. Their identifier key is defined by the colour matching functions and illuminant names along with the current shape and normalisation constant k such as: *CIE 1964 10 Degree Standard Observer, A, (360.0, 830.0, 10.0), None* Considering the above, one should be mindful that using similar colour matching functions and illuminant names but with different spectral data will lead to unexpected behaviour.

Notes

- Input colour matching functions and illuminant intervals are expected to be equal to 1 nm. If the illuminant data is not available at 1 nm interval, it needs to be interpolated using *CIE* recommendations: The method developed by *Sprague (1880)* should be used for interpolating functions having a uniformly spaced independent variable and a *Cubic Spline* method for non-uniformly spaced independent variable.

References

[ASTMInternational11]

Examples

```
>>> from colour import (CMFS, sd_CIE_standard_illuminant_A,
...     SpectralDistribution, SpectralShape)
>>> from colour.utilities import numpy_print_options
>>> cmfs = CMFS['CIE 1964 10 Degree Standard Observer']
>>> A = sd_CIE_standard_illuminant_A(cmfs.shape)
>>> with numpy_print_options(suppress=True):
...     tristimulus_weighting_factors_ASTME2022(
...         cmfs, A, SpectralShape(360, 830, 20))
...
array([[ -0.0002981...,  -0.0000317...,  -0.0013301...],
       [ -0.0087155...,  -0.0008915...,  -0.0407436...],
       [  0.0599679...,   0.0050203...,   0.2565018...],
       [  0.7734225...,   0.0779839...,   3.6965732...],
       [  1.9000905...,   0.3037005...,   9.7554195...],
       [  1.9707727...,   0.8552809...,  11.4867325...],
       [  0.7183623...,   2.1457000...,   6.7845806...],
       [  0.0426667...,   4.8985328...,   2.3208000...],
       [  1.5223302...,   9.6471138...,   0.7430671...],
       [  5.6770329...,  14.4609708...,   0.1958194...],
       [ 12.4451744...,  17.4742541...,   0.0051827...],
       [ 20.5535772...,  17.5838219...,  -0.0026512...],
       [ 25.3315384...,  14.8957035...,   0.        ...],
       [ 21.5711570...,  10.0796619...,   0.        ...],
       [ 12.1785817...,   5.0680655...,   0.        ...],
       [  4.6675746...,   1.8303239...,   0.        ...],
       [  1.3236117...,   0.5129694...,   0.        ...],
```

(continues on next page)

(continued from previous page)

```
       [  0.3175325...,   0.1230084...,   0.        ...],
       [  0.0746341...,   0.0290243...,   0.        ...],
       [  0.0182990...,   0.0071606...,   0.        ...],
       [  0.0047942...,   0.0018888...,   0.        ...],
       [  0.0013293...,   0.0005277...,   0.        ...],
       [  0.0004254...,   0.0001704...,   0.        ...],
       [  0.0000962...,   0.0000389...,   0.        ...]])
```

Integration

colour.colorimetry

sd_to_XYZ_integration(sd[, cmfs, illuminant, k])	Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.
multi_sds_to_XYZ_integration(msds[, cmfs, ...])	Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant.

colour.colorimetry.sd_to_XYZ_integration

colour.colorimetry.**sd_to_XYZ_integration**(*sd*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*)

Converts given spectral distribution to *CIE XYZ* tristimulus values using given colour matching functions and illuminant according to classical integration method.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.

Returns *CIE XYZ* tristimulus values.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[WS00f]

Examples

```
>>> from colour import (
...     CMFS, ILLUMINANTS_SDS, SpectralDistribution)
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> data = {
...     400: 0.0641,
...     420: 0.0645,
...     440: 0.0562,
...     460: 0.0537,
...     480: 0.0559,
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360,
...     620: 0.1511,
...     640: 0.1688,
...     660: 0.1996,
...     680: 0.2397,
...     700: 0.2852
... }
>>> sd = SpectralDistribution(data)
>>> illuminant = ILLUMINANTS_SDS['D65']
>>> sd_to_XYZ_integration(sd, cmfs, illuminant)
...
array([ 10.8401846...,   9.6837311...,   6.2120912...])
```

colour.colorimetry.multi_sds_to_XYZ_integration

colour.colorimetry.**multi_sds_to_XYZ_integration**(*msds*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *k=None*, *shape=SpectralShape(360, 780, 1)*)

Converts given multi-spectral distributions to *CIE XYZ* tristimulus values using given colour matching functions and illuminant. The multi-spectral distribution can be either a `colour.MultiSpectralDistributions` class instance or an *array_like* in which case the `shape` must be passed.

Parameters

- **msds** (`MultiSpectralDistributions` or `array_like`) – Multi-spectral distributions, if an *array_like* the wavelengths are expected to be in the last axis, e.g. for a 512x384 multi-spectral image with 77 bins, `msds` shape should be (384, 512, 77).

- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **k** (numeric, optional) – Normalisation constant k. For reflecting or transmitting object colours, k is chosen so that $Y = 100$ for objects for which the spectral reflectance factor $R(\lambda)$ of the object colour or the spectral transmittance factor $\tau(\lambda)$ of the object is equal to unity for all wavelengths. For self-luminous objects and illuminants, the constants k is usually chosen on the grounds of convenience. If, however, in the CIE 1931 standard colorimetric system, the Y value is required to be numerically equal to the absolute value of a photometric quantity, the constant, k, must be put equal to the numerical value of K_m, the maximum spectral luminous efficacy (which is equal to 683 $lm \cdot W^{-1}$) and $\Phi_\lambda(\lambda)$ must be the spectral concentration of the radiometric quantity corresponding to the photometric quantity required.
- **shape** (SpectralShape, optional) – Spectral shape of the multi-spectral distributions, cmfs and illuminant will be aligned to it.

Returns *CIE XYZ* tristimulus values, for a 512x384 multi-spectral image with 77 bins, the output shape will be (384, 512, 3).

Return type array_like

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- The code path using the *array_like* multi-spectral distributions produces results different to the code path using a colour.MultiSpectralDistributions class instance: the former favours execution speed by aligning the colour matching functions and illuminant to the given spectral shape while the latter favours precision by aligning the multi-spectral distributions to the colour matching functions.

References

[WS00f]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> shape = SpectralShape(400, 700, 60)
>>> D65 = ILLUMINANTS_SDS['D65']
>>> data = np.array([
...     [0.0137, 0.0159, 0.0096, 0.0111, 0.0179, 0.1057, 0.0433,
...      0.0258, 0.0248, 0.0186, 0.0310, 0.0473],
...     [0.0913, 0.3145, 0.2582, 0.0709, 0.2971, 0.4620, 0.2683,
...      0.0831, 0.1203, 0.1292, 0.1682, 0.3221],
...     [0.0152, 0.0842, 0.4139, 0.0220, 0.5630, 0.1918, 0.2373,
...      0.0430, 0.0054, 0.0079, 0.3719, 0.2268],
...     [0.0281, 0.0907, 0.2228, 0.1249, 0.2375, 0.5625, 0.0518,
...      0.3230, 0.0065, 0.4006, 0.0861, 0.3161],
...     [0.1918, 0.7103, 0.0041, 0.1817, 0.0024, 0.4209, 0.0118,
...      0.2302, 0.1860, 0.9404, 0.0041, 0.1124],
...     [0.0430, 0.0437, 0.3744, 0.0020, 0.5819, 0.0027, 0.0823,
```

(continues on next page)

(continued from previous page)

```
...      0.0081, 0.3625, 0.3213, 0.7849, 0.0024],
... ])
>>> msds = MultiSpectralDistributions(data, shape.range())
>>> multi_sds_to_XYZ_integration(msds, illuminant=D65, shape=shape)
...
array([[  7.5031039...,   3.9486887...,   8.4053425...],
       [ 26.9265732...,  15.0721676...,  28.7125195...],
       [ 16.7039284...,  28.2175962...,  25.6515081...],
       [ 11.5765075...,   8.6399161...,   6.5783111...],
       [ 18.7324243...,  35.0755847...,  30.1525320...],
       [ 45.1654991...,  39.6132308...,  43.6877410...],
       [  8.1765955...,  13.0937369...,  25.9477816...],
       [ 22.4668631...,  19.3095206...,   7.9654068...],
       [  6.5782666...,   2.5254216...,  11.0954901...],
       [ 43.9134487...,  27.9794809...,  11.731893 ...],
       [  8.5371481...,  19.7034075...,  17.7088101...],
       [ 23.9092001...,  26.2129391...,  30.6831630...]])
>>> msds = np.array([
...     [
...         [0.0137, 0.0913, 0.0152, 0.0281, 0.1918, 0.0430],
...         [0.0159, 0.3145, 0.0842, 0.0907, 0.7103, 0.0437],
...         [0.0096, 0.2582, 0.4139, 0.2228, 0.0041, 0.3744],
...         [0.0111, 0.0709, 0.0220, 0.1249, 0.1817, 0.0020],
...         [0.0179, 0.2971, 0.5630, 0.2375, 0.0024, 0.5819],
...         [0.1057, 0.4620, 0.1918, 0.5625, 0.4209, 0.0027],
...     ],
...     [
...         [0.0433, 0.2683, 0.2373, 0.0518, 0.0118, 0.0823],
...         [0.0258, 0.0831, 0.0430, 0.3230, 0.2302, 0.0081],
...         [0.0248, 0.1203, 0.0054, 0.0065, 0.1860, 0.3625],
...         [0.0186, 0.1292, 0.0079, 0.4006, 0.9404, 0.3213],
...         [0.0310, 0.1682, 0.3719, 0.0861, 0.0041, 0.7849],
...         [0.0473, 0.3221, 0.2268, 0.3161, 0.1124, 0.0024],
...     ],
... ])
>>> multi_sds_to_XYZ_integration(msds, illuminant=D65, shape=shape)
...
array([[[  7.1958378...,   3.8605390...,  10.1016398...],
        [ 25.5738615...,  14.7200581...,  34.8440007...],
        [ 17.5854414...,  28.5668344...,  30.1806687...],
        [ 11.3271912...,   8.4598177...,   7.9015758...],
        [ 19.6581831...,  35.5918480...,  35.1430220...],
        [ 45.8212491...,  39.2600939...,  51.7907710...]],

       [[  8.8287837...,  13.3870357...,  30.5702050...],
        [ 22.3324362...,  18.9560919...,   9.3952305...],
        [  6.6887212...,   2.5728891...,  13.2618778...],
        [ 41.8166227...,  27.1191979...,  14.2627944...],
        [  9.2414098...,  20.2056200...,  20.1992502...],
        [ 24.7830551...,  26.2221584...,  36.4430633...]]])
```

Spectral Bandpass Dependence Correction

colour

bandpass_correction(sd[, method])	Implements spectral bandpass dependence correction on given spectral distribution using given method.
BANDPASS_CORRECTION_METHODS	Supported spectral bandpass dependence correction methods.

colour.bandpass_correction

colour.**bandpass_correction**(*sd*, *method='Stearns 1988'*)

Implements spectral bandpass dependence correction on given spectral distribution using given method.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **method** (unicode, optional) – {'Stearns 1988', } Correction method.

Returns Spectral bandpass dependence corrected spectral distribution.

Return type *SpectralDistribution*

References

[SS88], [WRC12c]

Examples

```
>>> from colour import SpectralDistribution
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction(SpectralDistribution(data))
...
SpectralDistribution([[ 500.        ,    0.0646518...],
                      [ 520.        ,    0.0704293...],
                      [ 540.        ,    0.0769485...],
                      [ 560.        ,    0.0856928...],
                      [ 580.        ,    0.1129644...],
                      [ 600.        ,    0.1379256...]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
```

colour.BANDPASS_CORRECTION_METHODS

colour.**BANDPASS_CORRECTION_METHODS = CaseInsensitiveMapping({'Stearns 1988': ...})**
Supported spectral bandpass dependence correction methods.

BANDPASS_CORRECTION_METHODS [CaseInsensitiveMapping] **{'Stearns 1988', }**

Stearns and Stearns (1988)

colour.colorimetry

bandpass_correction_Stearns1988(sd)	Implements spectral bandpass dependence correction on given spectral distribution using *Stearns and Stearns (1988)* method.

colour.colorimetry.bandpass_correction_Stearns1988

colour.colorimetry.**bandpass_correction_Stearns1988**(*sd*)
Implements spectral bandpass dependence correction on given spectral distribution using *Stearns and Stearns (1988)* method.

Parameters **sd** (SpectralDistribution) – Spectral distribution.

Returns Spectral bandpass dependence corrected spectral distribution.

Return type *SpectralDistribution*

References

[SS88], [WRC12c]

Examples

```
>>> from colour import SpectralDistribution
>>> from colour.utilities import numpy_print_options
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> with numpy_print_options(suppress=True):
...     bandpass_correction_Stearns1988(SpectralDistribution(data))
...
SpectralDistribution([[ 500.        ,    0.0646518...],
                      [ 520.        ,    0.0704293...],
                      [ 540.        ,    0.0769485...],
                      [ 560.        ,    0.0856928...],
                      [ 580.        ,    0.1129644...],
                      [ 600.        ,    0.1379256...]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
```

Colour Matching Functions

colour.colorimetry

LMS_ConeFundamentals([data, domain, labels])	Implements support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.
RGB_ColourMatchingFunctions([data, domain, ...])	Implements support for the *CIE RGB* colour matching functions.
XYZ_ColourMatchingFunctions([data, domain, ...])	Implements support for the *CIE* Standard Observers *XYZ* colour matching functions.

colour.colorimetry.LMS_ConeFundamentals

class colour.colorimetry.**LMS_ConeFundamentals**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for the Stockman and Sharpe *LMS* cone fundamentals colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the Values to initialise the multiple colour.SpectralDistribution colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to colour.colorimetry.LMS_ConeFundamentals.labels attribute value.

colour.colorimetry.RGB_ColourMatchingFunctions

class colour.colorimetry.**RGB_ColourMatchingFunctions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for the *CIE RGB* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.SpectralDistribution class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.SpectralDistribution class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.SpectralDistribution class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to colour.colorimetry.RGB_ColourMatchingFunctions.labels attribute value.

colour.colorimetry.XYZ_ColourMatchingFunctions

class colour.colorimetry.**XYZ_ColourMatchingFunctions**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.colorimetry.spectrum.MultiSpectralDistributions

Implements support for the *CIE* Standard Observers *XYZ* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the `colour.SpectralDistribution` class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to `colour.colorimetry.XYZ_ColourMatchingFunctions.labels` attribute value.

Dataset

`colour`

`CMFS`	Aggregated colour matching functions.
`LMS_CMFS`	*LMS* colour matching functions.
`RGB_CMFS`	*CIE RGB* colour matching functions.
`STANDARD_OBSERVERS_CMFS`	*CIE* Standard Observers *XYZ* colour matching functions.

colour.CMFS

`colour.`**`CMFS = CaseInsensitiveMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman & Sh`**

Aggregated colour matching functions.

References

[Bro09], [CVRd], [CVRe], [CVRf], [SS00], [CVRg], [Mac10]

CMFS [CaseInsensitiveMapping] **{'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs', 'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer', 'CIE 2012 2 Degree Standard Observer', 'CIE 2012 10 Degree Standard Observer'}**

colour.LMS_CMFS

`colour.`**`LMS_CMFS = CaseInsensitiveMapping({'Stockman & Sharpe 2 Degree Cone Fundamentals': ..., 'Stockman`**

LMS colour matching functions.

References

[SS00], [Mac10]

LMS_CMFS [CaseInsensitiveMapping] {'Stockman & Sharpe 2 Degree Cone Fundamentals', 'Stockman & Sharpe 10 Degree Cone Fundamentals', 'Smith & Pokorny 1975 Normal Trichromats'}

colour.RGB_CMFS

colour.**RGB_CMFS = CaseInsensitiveMapping({'Wright & Guild 1931 2 Degree RGB CMFs': ..., 'Stiles & Burch 1**
CIE RGB colour matching functions.

References

[Bro09], [CVRf], [CVRg]

RGB_CMFS [CaseInsensitiveMapping] **{'Wright & Guild 1931 2 Degree RGB CMFs', 'Stiles & Burch 1955 2 Degree RGB CMFs', 'Stiles & Burch 1959 10 Degree RGB CMFs'}**

colour.STANDARD_OBSERVERS_CMFS

colour.**STANDARD_OBSERVERS_CMFS = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE**
CIE Standard Observers *XYZ* colour matching functions.

References

[CVRd], [CVRe]

STANDARD_OBSERVERS_CMFS [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer', 'CIE 2012 2 Degree Standard Observer', 'CIE 2012 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

Colour Matching Functions Transformations

Ancillary Objects

colour.colorimetry

RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs(...)	Converts *Wright & Guild 1931 2 Degree RGB CMFs* colour matching functions into the *CIE 1931 2 Degree Standard Observer* colour matching functions.
RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs(...)	Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *CIE 1964 10 Degree Standard Observer* colour matching functions.
RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(...)	Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral sensitivity functions.
LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(...)	Converts *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 2 Degree Standard Observer* colour matching functions.

Continued on next page

Table 83 – continued from previous page

LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(...)	Converts *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 10 Degree Standard Observer* colour matching functions.

colour.colorimetry.RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs

colour.colorimetry.**RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs**(*wavelength*)

Converts *Wright & Guild 1931 2 Degree RGB CMFs* colour matching functions into the *CIE 1931 2 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 1931 2 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 1931 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[WS00i]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs(700)
array([ 0.0113577...,  0.004102  ,  0.        ])
```

colour.colorimetry.RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs

colour.colorimetry.**RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs**(*wavelength*)

Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *CIE 1964 10 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 1964 10 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 1964 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[WS00m]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs(700)
array([ 0.0096432...,  0.0037526..., -0.0000041...])
```

colour.colorimetry.RGB_10_degree_cmfs_to_LMS_10_degree_cmfs

colour.colorimetry.**RGB_10_degree_cmfs_to_LMS_10_degree_cmfs**(*wavelength*)

Converts *Stiles & Burch 1959 10 Degree RGB CMFs* colour matching functions into the *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral sensitivity functions.

Parameters **wavelength** (numeric or array_like) – Wavelength λ in nm.

Returns *Stockman & Sharpe 10 Degree Cone Fundamentals* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *Stockman & Sharpe 10 Degree Cone Fundamentals* already exists, this definition is intended for educational purpose.

References

[CIET13606]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     RGB_10_degree_cmfs_to_LMS_10_degree_cmfs(700)
array([ 0.0052860...,  0.0003252...,  0.        ])
```

colour.colorimetry.LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs

colour.colorimetry.**LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs**(*wavelength*)

Converts *Stockman & Sharpe 2 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 2 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 2012 2 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 2012 2 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[CVRb]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs(700)
array([ 0.0109677...,  0.0041959...,  0.        ])
```

colour.colorimetry.LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs

colour.colorimetry.**LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs**(*wavelength*)

Converts *Stockman & Sharpe 10 Degree Cone Fundamentals* colour matching functions into the *CIE 2012 10 Degree Standard Observer* colour matching functions.

Parameters **wavelength** (numeric or array_like) – Wavelength λ in nm.

Returns *CIE 2012 10 Degree Standard Observer* spectral tristimulus values.

Return type ndarray

Notes

- Data for the *CIE 2012 10 Degree Standard Observer* already exists, this definition is intended for educational purpose.

References

[CVRa]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs(700)
array([ 0.0098162...,  0.0037761...,  0.        ])
```

Illuminants and Light Sources

Dataset

colour

ILLUMINANTS	Aggregated *CIE* illuminants chromaticity coordinates.
ILLUMINANTS_SDS	*CIE* illuminants spectral distributions.
HUNTERLAB_ILLUMINANTS	Aggregated *Hunter L,a,b* illuminant datasets.
LIGHT_SOURCES	Aggregated light sources chromaticity coordinates.
LIGHT_SOURCES_SDS	Aggregated light sources spectral distributions.

colour.ILLUMINANTS

colour.**ILLUMINANTS = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 Deg**

Aggregated *CIE* illuminants chromaticity coordinates.

> **Warning:** *DCI-P3* illuminant has no associated spectral distribution. *DCI* has no official reference spectral measurement for this whitepoint. The closest matching spectral distribution is *Kinoton 75P* projector.

Notes

CIE Illuminant D Series D60 illuminant chromaticity coordinates were computed as follows:

```
CCT = 6000 * 1.4388 / 1.438
xy = colour.temperature.CCT_to_xy_CIE_D(CCT)

sd = colour.sd_CIE_illuminant_D_series(xy)
sd.interpolator = colour.LinearInterpolator
colour.XYZ_to_xy(
    colour.sd_to_XYZ(
        sd, colour.CMFS['CIE 1964 10 Degree Standard Observer']) / 100.0)
```

- *CIE Illuminant D Series D50* illuminant and *CIE Standard Illuminant D Series D65* chromaticity coordinates are rounded to 4 decimals as given in the typical RGB colourspaces litterature. Their chromaticity coordinates as given in [CIET14804d] are (0.34567, 0.35851) and (0.31272, 0.32903) respectively.
- *CIE* illuminants with chromaticity coordinates not defined in the reference [Wik06b] have been calculated using their correlated colour temperature and colour.temperature.CCT_to_xy_CIE_D() colour.sd_CIE_illuminant_D_series() and / or colour.sd_to_XYZ() definitions.
- *ICC D50* chromaticity coordinates were computed with colour.XYZ_to_xy() definition from the *CIE XYZ* tristimulus values as given by *ICC*: [96.42, 100.00, 82.49].

References

[CIET14804d], [DigitalCInitiatives07], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14
[Wik06b]

ILLUMINANTS [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.ILLUMINANTS_SDS

colour.**ILLUMINANTS_SDS = CaseInsensitiveMapping({'A': ..., 'B': ..., 'C': ..., 'D50': ..., 'D55': ..., '**
CIE illuminants spectral distributions.

Notes

CIE Illuminant D Series D60 illuminant spectral distribution was computed as follows:

```
CCT = 6000 * 1.4388 / 1.438
xy = colour.temperature.CCT_to_xy_CIE_D(CCT)

sd = colour.sd_CIE_illuminant_D_series(xy)
```

References

[CSH+18], [CIE04], [CIE]

ILLUMINANTS_SDS : CaseInsensitiveMapping

colour.HUNTERLAB_ILLUMINANTS

colour.**HUNTERLAB_ILLUMINANTS = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1**
Aggregated *Hunter L,a,b* illuminant datasets.

References

[Hun08a], [Hun08b]

HUNTERLAB_ILLUMINANTS [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.LIGHT_SOURCES

colour.**LIGHT_SOURCES = CaseInsensitiveMapping({'CIE 1931 2 Degree Standard Observer': ..., 'CIE 1964 10 D**

Aggregated light sources chromaticity coordinates.

LIGHT_SOURCES [CaseInsensitiveMapping] **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**

Aliases:

- 'cie_2_1931': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 10 Degree Standard Observer'

colour.LIGHT_SOURCES_SDS

colour.**LIGHT_SOURCES_SDS = CaseInsensitiveMapping({'Natural': ..., 'Philips TL-84': ..., 'SA': ..., 'SC'**

Aggregated light sources spectral distributions.

LIGHT_SOURCES_SDS : CaseInsensitiveMapping

Dominant Wavelength and Purity

colour

dominant_wavelength(xy, xy_n[, cmfs, inverse])	Returns the *dominant wavelength* λ_d for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.
complementary_wavelength(xy, xy_n[, cmfs])	Returns the *complementary wavelength* λ_c for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.
excitation_purity(xy, xy_n[, cmfs])	Returns the *excitation purity* P_e for given colour stimulus xy.
colorimetric_purity(xy, xy_n[, cmfs])	Returns the *colorimetric purity* P_c for given colour stimulus xy.

colour.dominant_wavelength

colour.**dominant_wavelength**(*xy*, *xy_n*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *inverse=False*)

Returns the *dominant wavelength* λ_d for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.

In the eventuality where the xy_wl first intersection coordinates are on the line of purples, the *complementary wavelength* will be computed in lieu.

The *complementary wavelength* is indicated by a negative sign and the xy_{cw} second intersection coordinates which are set by default to the same value than xy_wl first intersection coordinates will be set to the *complementary dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (array_like) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

- **inverse** (bool, optional) – Inverse the computation direction to retrieve the *complementary wavelength*.

Returns *Dominant wavelength*, first intersection point *CIE xy* chromaticity coordinates, second intersection point *CIE xy* chromaticity coordinates.

Return type tuple

References

[CIET14804b], [Erdb]

Examples

Dominant wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(dominant_wavelength(xy, xy_n, cmfs))
(array(616...),
 array([ 0.6835474...,  0.3162840...]),
 array([ 0.6835474...,  0.3162840...]))
```

Complementary dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.37605506, 0.24452225])
>>> pprint(dominant_wavelength(xy, xy_n, cmfs))
(array(-509.0),
 array([ 0.4572314...,  0.1362814...]),
 array([ 0.0104096...,  0.7320745...]))
```

colour.complementary_wavelength

colour.**complementary_wavelength**(*xy*, *xy_n*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *complementary wavelength* λ_c for given colour stimulus xy and the related xy_wl first and xy_{cw} second intersection coordinates with the spectral locus.

In the eventuality where the xy_wl first intersection coordinates are on the line of purples, the *dominant wavelength* will be computed in lieu.

The *dominant wavelength* is indicated by a negative sign and the xy_{cw} second intersection coordinates which are set by default to the same value than xy_wl first intersection coordinates will be set to the *dominant wavelength* intersection coordinates with the spectral locus.

Parameters

- **xy** (array_like) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Complementary wavelength*, first intersection point *CIE xy* chromaticity coordinates, second intersection point *CIE xy* chromaticity coordinates.

Return type tuple

References

[CIET14804b], [Erdb]

Examples

Complementary wavelength computation:

```
>>> from pprint import pprint
>>> xy = np.array([0.37605506, 0.24452225])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> pprint(complementary_wavelength(xy, xy_n, cmfs))
(array(509.0),
 array([ 0.0104096...,  0.7320745...]),
 array([ 0.0104096...,  0.7320745...]))
```

Dominant wavelength is returned if the first intersection is located on the line of purples:

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> pprint(complementary_wavelength(xy, xy_n, cmfs))
(array(492.0),
 array([ 0.0364795 ,  0.3384712...]),
 array([ 0.0364795 ,  0.3384712...]))
```

colour.excitation_purity

colour.**excitation_purity**(*xy*, *xy_n*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *excitation purity* P_e for given colour stimulus xy.

Parameters

- **xy** (array_like) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Excitation purity* P_e.

Return type numeric or array_like

References

[CIET14804b], [Erdb]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> excitation_purity(xy, xy_n, cmfs)
0.6228856...
```

colour.colorimetric_purity

colour.**colorimetric_purity**(*xy*, *xy_n*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *colorimetric purity* P_c for given colour stimulus xy.

Parameters

- **xy** (array_like) – Colour stimulus *CIE xy* chromaticity coordinates.
- **xy_n** (array_like) – Achromatic stimulus *CIE xy* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *Colorimetric purity* P_c.

Return type numeric or array_like

References

[CIET14804b], [Erdb]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_n = np.array([0.31270000, 0.32900000])
>>> cmfs = CMFS['CIE 1931 2 Degree Standard Observer']
>>> colorimetric_purity(xy, xy_n, cmfs)
0.6135828...
```

Luminous Efficiency Functions

colour

luminous_efficacy(sd[, lef])	Returns the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.
luminous_efficiency(sd[, lef])	Returns the *luminous efficiency* of given spectral distribution using given luminous efficiency function.
luminous_flux(sd[, lef, K_m])	Returns the *luminous flux* for given spectral distribution using given luminous efficiency function.
sd_mesopic_luminous_efficiency_function(Lp)	Returns the mesopic luminous efficiency function $V_m(\lambda)$ for given photopic luminance L_p.

colour.luminous_efficacy

colour.**luminous_efficacy**(*sd*, *lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...)*)

Returns the *luminous efficacy* in $lm \cdot W^{-1}$ of given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (SpectralDistribution) – test spectral distribution
- **lef** (SpectralDistribution, optional) – $V(\lambda)$ luminous efficiency function.

Returns Luminous efficacy in $lm \cdot W^{-1}$.

Return type numeric

References

[Wik05d]

Examples

```
>>> from colour import LIGHT_SOURCES_SDS
>>> sd = LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> luminous_efficacy(sd)
136.2170803...
```

colour.luminous_efficiency

colour.**luminous_efficiency**(*sd*, *lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...)*)

Returns the *luminous efficiency* of given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (SpectralDistribution) – test spectral distribution
- **lef** (SpectralDistribution, optional) – $V(\lambda)$ luminous efficiency function.

Returns Luminous efficiency.

Return type numeric

References

[Wik03c]

Examples

```
>>> from colour import LIGHT_SOURCES_SDS
>>> sd = LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> luminous_efficiency(sd)
0.1994393...
```

colour.luminous_flux

colour.**luminous_flux**(*sd*, *lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...)*, *K_m=683.0*)

Returns the *luminous flux* for given spectral distribution using given luminous efficiency function.

Parameters

- **sd** (SpectralDistribution) – test spectral distribution
- **lef** (SpectralDistribution, optional) – $V(\lambda)$ luminous efficiency function.
- **K_m** (numeric, optional) – $lm \cdot W^{-1}$ maximum photopic luminous efficiency

Returns Luminous flux.

Return type numeric

References

[Wik03c]

Examples

```
>>> from colour import LIGHT_SOURCES_SDS
>>> sd = LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> luminous_flux(sd)
23807.6555273...
```

colour.sd_mesopic_luminous_efficiency_function

colour.**sd_mesopic_luminous_efficiency_function**(*Lp*, *source='Blue Heavy'*, *method='MOVE'*, *photopic_lef=SpectralDistribution(name='CIE 1924 Photopic Standard Observer', ...)*, *scotopic_lef=SpectralDistribution(name='CIE 1951 Scotopic Standard Observer', ...)*)

Returns the mesopic luminous efficiency function $V_m(\lambda)$ for given photopic luminance L_p.

Parameters

- **Lp** (numeric) – Photopic luminance L_p.
- **source** (unicode, optional) – **{'Blue Heavy', 'Red Heavy'}**, Light source colour temperature.
- **method** (unicode, optional) – **{'MOVE', 'LRC'}**, Method to calculate the weighting factor.
- **photopic_lef** (SpectralDistribution, optional) – $V(\lambda)$ photopic luminous efficiency function.
- **scotopic_lef** (SpectralDistribution, optional) – $V'(\lambda)$ scotopic luminous efficiency function.

Returns Mesopic luminous efficiency function $V_m(\lambda)$.

Return type *SpectralDistribution*

References

[Wik05e]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_mesopic_luminous_efficiency_function(0.2)
SpectralDistribution([[ 380.        ,    0.000424 ...],
                      [ 381.        ,    0.0004781...],
                      [ 382.        ,    0.0005399...],
                      [ 383.        ,    0.0006122...],
                      [ 384.        ,    0.0006961...],
                      [ 385.        ,    0.0007929...],
```

(continues on next page)

(continued from previous page)

```
                    [ 386.        ,    0.000907 ...],
                    [ 387.        ,    0.0010389...],
                    [ 388.        ,    0.0011923...],
                    [ 389.        ,    0.0013703...],
                    [ 390.        ,    0.0015771...],
                    [ 391.        ,    0.0018167...],
                    [ 392.        ,    0.0020942...],
                    [ 393.        ,    0.0024160...],
                    [ 394.        ,    0.0027888...],
                    [ 395.        ,    0.0032196...],
                    [ 396.        ,    0.0037222...],
                    [ 397.        ,    0.0042957...],
                    [ 398.        ,    0.0049531...],
                    [ 399.        ,    0.0057143...],
                    [ 400.        ,    0.0065784...],
                    [ 401.        ,    0.0075658...],
                    [ 402.        ,    0.0086912...],
                    [ 403.        ,    0.0099638...],
                    [ 404.        ,    0.0114058...],
                    [ 405.        ,    0.0130401...],
                    [ 406.        ,    0.0148750...],
                    [ 407.        ,    0.0169310...],
                    [ 408.        ,    0.0192211...],
                    [ 409.        ,    0.0217511...],
                    [ 410.        ,    0.0245342...],
                    [ 411.        ,    0.0275773...],
                    [ 412.        ,    0.0309172...],
                    [ 413.        ,    0.0345149...],
                    [ 414.        ,    0.0383998...],
                    [ 415.        ,    0.0425744...],
                    [ 416.        ,    0.0471074...],
                    [ 417.        ,    0.0519322...],
                    [ 418.        ,    0.0570541...],
                    [ 419.        ,    0.0625466...],
                    [ 420.        ,    0.0683463...],
                    [ 421.        ,    0.0745255...],
                    [ 422.        ,    0.0809440...],
                    [ 423.        ,    0.0877344...],
                    [ 424.        ,    0.0948915...],
                    [ 425.        ,    0.1022731...],
                    [ 426.        ,    0.109877 ...],
                    [ 427.        ,    0.1178421...],
                    [ 428.        ,    0.1260316...],
                    [ 429.        ,    0.1343772...],
                    [ 430.        ,    0.143017 ...],
                    [ 431.        ,    0.1518128...],
                    [ 432.        ,    0.1608328...],
                    [ 433.        ,    0.1700088...],
                    [ 434.        ,    0.1792726...],
                    [ 435.        ,    0.1886934...],
                    [ 436.        ,    0.1982041...],
                    [ 437.        ,    0.2078032...],
                    [ 438.        ,    0.2174184...],
                    [ 439.        ,    0.2271147...],
                    [ 440.        ,    0.2368196...],
                    [ 441.        ,    0.2464623...],
                    [ 442.        ,    0.2561153...],
                    [ 443.        ,    0.2657160...],
                    [ 444.        ,    0.2753387...],
                    [ 445.        ,    0.2848520...],
                    [ 446.        ,    0.2944648...],
```

(continues on next page)

(continued from previous page)

```
                       [ 447.        ,    0.3034902...],
                       [ 448.        ,    0.3132347...],
                       [ 449.        ,    0.3223257...],
                       [ 450.        ,    0.3314513...],
                       [ 451.        ,    0.3406129...],
                       [ 452.        ,    0.3498117...],
                       [ 453.        ,    0.3583617...],
                       [ 454.        ,    0.3676377...],
                       [ 455.        ,    0.3762670...],
                       [ 456.        ,    0.3849392...],
                       [ 457.        ,    0.3936540...],
                       [ 458.        ,    0.4024077...],
                       [ 459.        ,    0.4111965...],
                       [ 460.        ,    0.4193298...],
                       [ 461.        ,    0.4281803...],
                       [ 462.        ,    0.4363804...],
                       [ 463.        ,    0.4453117...],
                       [ 464.        ,    0.4542949...],
                       [ 465.        ,    0.4626509...],
                       [ 466.        ,    0.4717570...],
                       [ 467.        ,    0.4809300...],
                       [ 468.        ,    0.4901776...],
                       [ 469.        ,    0.4995075...],
                       [ 470.        ,    0.5096145...],
                       [ 471.        ,    0.5191293...],
                       [ 472.        ,    0.5294259...],
                       [ 473.        ,    0.5391316...],
                       [ 474.        ,    0.5496217...],
                       [ 475.        ,    0.5602103...],
                       [ 476.        ,    0.5702197...],
                       [ 477.        ,    0.5810207...],
                       [ 478.        ,    0.5919093...],
                       [ 479.        ,    0.6028683...],
                       [ 480.        ,    0.6138806...],
                       [ 481.        ,    0.6249373...],
                       [ 482.        ,    0.6360619...],
                       [ 483.        ,    0.6465989...],
                       [ 484.        ,    0.6579538...],
                       [ 485.        ,    0.6687841...],
                       [ 486.        ,    0.6797939...],
                       [ 487.        ,    0.6909887...],
                       [ 488.        ,    0.7023827...],
                       [ 489.        ,    0.7133032...],
                       [ 490.        ,    0.7244513...],
                       [ 491.        ,    0.7358470...],
                       [ 492.        ,    0.7468118...],
                       [ 493.        ,    0.7580294...],
                       [ 494.        ,    0.7694964...],
                       [ 495.        ,    0.7805225...],
                       [ 496.        ,    0.7917805...],
                       [ 497.        ,    0.8026123...],
                       [ 498.        ,    0.8130793...],
                       [ 499.        ,    0.8239297...],
                       [ 500.        ,    0.8352251...],
                       [ 501.        ,    0.8456342...],
                       [ 502.        ,    0.8564818...],
                       [ 503.        ,    0.8676921...],
                       [ 504.        ,    0.8785021...],
                       [ 505.        ,    0.8881489...],
                       [ 506.        ,    0.8986405...],
                       [ 507.        ,    0.9079322...],
```

(continues on next page)

(continued from previous page)

```
                    [ 508.        ,    0.9174255...],
                    [ 509.        ,    0.9257739...],
                    [ 510.        ,    0.9350656...],
                    [ 511.        ,    0.9432365...],
                    [ 512.        ,    0.9509063...],
                    [ 513.        ,    0.9586931...],
                    [ 514.        ,    0.9658413...],
                    [ 515.        ,    0.9722825...],
                    [ 516.        ,    0.9779924...],
                    [ 517.        ,    0.9836106...],
                    [ 518.        ,    0.9883465...],
                    [ 519.        ,    0.9920964...],
                    [ 520.        ,    0.9954436...],
                    [ 521.        ,    0.9976202...],
                    [ 522.        ,    0.9993457...],
                    [ 523.        ,    1.       ...],
                    [ 524.        ,    0.9996498...],
                    [ 525.        ,    0.9990487...],
                    [ 526.        ,    0.9975356...],
                    [ 527.        ,    0.9957615...],
                    [ 528.        ,    0.9930143...],
                    [ 529.        ,    0.9899559...],
                    [ 530.        ,    0.9858741...],
                    [ 531.        ,    0.9814453...],
                    [ 532.        ,    0.9766885...],
                    [ 533.        ,    0.9709363...],
                    [ 534.        ,    0.9648947...],
                    [ 535.        ,    0.9585832...],
                    [ 536.        ,    0.952012 ...],
                    [ 537.        ,    0.9444916...],
                    [ 538.        ,    0.9367089...],
                    [ 539.        ,    0.9293506...],
                    [ 540.        ,    0.9210429...],
                    [ 541.        ,    0.9124772...],
                    [ 542.        ,    0.9036604...],
                    [ 543.        ,    0.8945958...],
                    [ 544.        ,    0.8845999...],
                    [ 545.        ,    0.8750500...],
                    [ 546.        ,    0.8659457...],
                    [ 547.        ,    0.8559224...],
                    [ 548.        ,    0.8456846...],
                    [ 549.        ,    0.8352499...],
                    [ 550.        ,    0.8253229...],
                    [ 551.        ,    0.8152079...],
                    [ 552.        ,    0.8042205...],
                    [ 553.        ,    0.7944209...],
                    [ 554.        ,    0.7837466...],
                    [ 555.        ,    0.7735680...],
                    [ 556.        ,    0.7627808...],
                    [ 557.        ,    0.7522710...],
                    [ 558.        ,    0.7417549...],
                    [ 559.        ,    0.7312909...],
                    [ 560.        ,    0.7207983...],
                    [ 561.        ,    0.7101939...],
                    [ 562.        ,    0.6996362...],
                    [ 563.        ,    0.6890656...],
                    [ 564.        ,    0.6785599...],
                    [ 565.        ,    0.6680593...],
                    [ 566.        ,    0.6575697...],
                    [ 567.        ,    0.6471578...],
                    [ 568.        ,    0.6368208...],
```

(continues on next page)

(continued from previous page)

```
                       [ 569.        ,    0.6264871...],
                       [ 570.        ,    0.6161541...],
                       [ 571.        ,    0.6058896...],
                       [ 572.        ,    0.5957000...],
                       [ 573.        ,    0.5855937...],
                       [ 574.        ,    0.5754412...],
                       [ 575.        ,    0.5653883...],
                       [ 576.        ,    0.5553742...],
                       [ 577.        ,    0.5454680...],
                       [ 578.        ,    0.5355972...],
                       [ 579.        ,    0.5258267...],
                       [ 580.        ,    0.5160152...],
                       [ 581.        ,    0.5062322...],
                       [ 582.        ,    0.4965595...],
                       [ 583.        ,    0.4868746...],
                       [ 584.        ,    0.4773299...],
                       [ 585.        ,    0.4678028...],
                       [ 586.        ,    0.4583704...],
                       [ 587.        ,    0.4489722...],
                       [ 588.        ,    0.4397606...],
                       [ 589.        ,    0.4306131...],
                       [ 590.        ,    0.4215446...],
                       [ 591.        ,    0.4125681...],
                       [ 592.        ,    0.4037550...],
                       [ 593.        ,    0.3950359...],
                       [ 594.        ,    0.3864104...],
                       [ 595.        ,    0.3778777...],
                       [ 596.        ,    0.3694405...],
                       [ 597.        ,    0.3611074...],
                       [ 598.        ,    0.3528596...],
                       [ 599.        ,    0.3447056...],
                       [ 600.        ,    0.3366470...],
                       [ 601.        ,    0.3286917...],
                       [ 602.        ,    0.3208410...],
                       [ 603.        ,    0.3130808...],
                       [ 604.        ,    0.3054105...],
                       [ 605.        ,    0.2978225...],
                       [ 606.        ,    0.2903027...],
                       [ 607.        ,    0.2828727...],
                       [ 608.        ,    0.2755311...],
                       [ 609.        ,    0.2682900...],
                       [ 610.        ,    0.2611478...],
                       [ 611.        ,    0.2541176...],
                       [ 612.        ,    0.2471885...],
                       [ 613.        ,    0.2403570...],
                       [ 614.        ,    0.2336057...],
                       [ 615.        ,    0.2269379...],
                       [ 616.        ,    0.2203527...],
                       [ 617.        ,    0.2138465...],
                       [ 618.        ,    0.2073946...],
                       [ 619.        ,    0.2009789...],
                       [ 620.        ,    0.1945818...],
                       [ 621.        ,    0.1881943...],
                       [ 622.        ,    0.1818226...],
                       [ 623.        ,    0.1754987...],
                       [ 624.        ,    0.1692476...],
                       [ 625.        ,    0.1630876...],
                       [ 626.        ,    0.1570257...],
                       [ 627.        ,    0.151071 ...],
                       [ 628.        ,    0.1452469...],
                       [ 629.        ,    0.1395845...],
```

(continues on next page)

(continued from previous page)

```
                       [ 630.        ,    0.1341087...],
                       [ 631.        ,    0.1288408...],
                       [ 632.        ,    0.1237666...],
                       [ 633.        ,    0.1188631...],
                       [ 634.        ,    0.1141075...],
                       [ 635.        ,    0.1094766...],
                       [ 636.        ,    0.1049613...],
                       [ 637.        ,    0.1005679...],
                       [ 638.        ,    0.0962924...],
                       [ 639.        ,    0.0921296...],
                       [ 640.        ,    0.0880778...],
                       [ 641.        ,    0.0841306...],
                       [ 642.        ,    0.0802887...],
                       [ 643.        ,    0.0765559...],
                       [ 644.        ,    0.0729367...],
                       [ 645.        ,    0.0694345...],
                       [ 646.        ,    0.0660491...],
                       [ 647.        ,    0.0627792...],
                       [ 648.        ,    0.0596278...],
                       [ 649.        ,    0.0565970...],
                       [ 650.        ,    0.0536896...],
                       [ 651.        ,    0.0509068...],
                       [ 652.        ,    0.0482444...],
                       [ 653.        ,    0.0456951...],
                       [ 654.        ,    0.0432510...],
                       [ 655.        ,    0.0409052...],
                       [ 656.        ,    0.0386537...],
                       [ 657.        ,    0.0364955...],
                       [ 658.        ,    0.0344285...],
                       [ 659.        ,    0.0324501...],
                       [ 660.        ,    0.0305579...],
                       [ 661.        ,    0.0287496...],
                       [ 662.        ,    0.0270233...],
                       [ 663.        ,    0.0253776...],
                       [ 664.        ,    0.0238113...],
                       [ 665.        ,    0.0223226...],
                       [ 666.        ,    0.0209086...],
                       [ 667.        ,    0.0195688...],
                       [ 668.        ,    0.0183056...],
                       [ 669.        ,    0.0171216...],
                       [ 670.        ,    0.0160192...],
                       [ 671.        ,    0.0149986...],
                       [ 672.        ,    0.0140537...],
                       [ 673.        ,    0.0131784...],
                       [ 674.        ,    0.0123662...],
                       [ 675.        ,    0.0116107...],
                       [ 676.        ,    0.0109098...],
                       [ 677.        ,    0.0102587...],
                       [ 678.        ,    0.0096476...],
                       [ 679.        ,    0.0090665...],
                       [ 680.        ,    0.0085053...],
                       [ 681.        ,    0.0079567...],
                       [ 682.        ,    0.0074229...],
                       [ 683.        ,    0.0069094...],
                       [ 684.        ,    0.0064213...],
                       [ 685.        ,    0.0059637...],
                       [ 686.        ,    0.0055377...],
                       [ 687.        ,    0.0051402...],
                       [ 688.        ,    0.00477  ...],
                       [ 689.        ,    0.0044263...],
                       [ 690.        ,    0.0041081...],
```

(continues on next page)

(continued from previous page)

```
                       [ 691.        ,    0.0038149...],
                       [ 692.        ,    0.0035456...],
                       [ 693.        ,    0.0032984...],
                       [ 694.        ,    0.0030718...],
                       [ 695.        ,    0.0028639...],
                       [ 696.        ,    0.0026738...],
                       [ 697.        ,    0.0025000...],
                       [ 698.        ,    0.0023401...],
                       [ 699.        ,    0.0021918...],
                       [ 700.        ,    0.0020526...],
                       [ 701.        ,    0.0019207...],
                       [ 702.        ,    0.001796 ...],
                       [ 703.        ,    0.0016784...],
                       [ 704.        ,    0.0015683...],
                       [ 705.        ,    0.0014657...],
                       [ 706.        ,    0.0013702...],
                       [ 707.        ,    0.001281 ...],
                       [ 708.        ,    0.0011976...],
                       [ 709.        ,    0.0011195...],
                       [ 710.        ,    0.0010464...],
                       [ 711.        ,    0.0009776...],
                       [ 712.        ,    0.0009131...],
                       [ 713.        ,    0.0008525...],
                       [ 714.        ,    0.0007958...],
                       [ 715.        ,    0.0007427...],
                       [ 716.        ,    0.0006929...],
                       [ 717.        ,    0.0006462...],
                       [ 718.        ,    0.0006026...],
                       [ 719.        ,    0.0005619...],
                       [ 720.        ,    0.0005240...],
                       [ 721.        ,    0.0004888...],
                       [ 722.        ,    0.0004561...],
                       [ 723.        ,    0.0004255...],
                       [ 724.        ,    0.0003971...],
                       [ 725.        ,    0.0003704...],
                       [ 726.        ,    0.0003455...],
                       [ 727.        ,    0.0003221...],
                       [ 728.        ,    0.0003001...],
                       [ 729.        ,    0.0002796...],
                       [ 730.        ,    0.0002604...],
                       [ 731.        ,    0.0002423...],
                       [ 732.        ,    0.0002254...],
                       [ 733.        ,    0.0002095...],
                       [ 734.        ,    0.0001947...],
                       [ 735.        ,    0.0001809...],
                       [ 736.        ,    0.0001680...],
                       [ 737.        ,    0.0001560...],
                       [ 738.        ,    0.0001449...],
                       [ 739.        ,    0.0001345...],
                       [ 740.        ,    0.0001249...],
                       [ 741.        ,    0.0001159...],
                       [ 742.        ,    0.0001076...],
                       [ 743.        ,    0.0000999...],
                       [ 744.        ,    0.0000927...],
                       [ 745.        ,    0.0000862...],
                       [ 746.        ,    0.0000801...],
                       [ 747.        ,    0.0000745...],
                       [ 748.        ,    0.0000693...],
                       [ 749.        ,    0.0000646...],
                       [ 750.        ,    0.0000602...],
                       [ 751.        ,    0.0000561...],
```

(continues on next page)

(continued from previous page)

```
                    [ 752.        ,    0.0000523...],
                    [ 753.        ,    0.0000488...],
                    [ 754.        ,    0.0000456...],
                    [ 755.        ,    0.0000425...],
                    [ 756.        ,    0.0000397...],
                    [ 757.        ,    0.0000370...],
                    [ 758.        ,    0.0000346...],
                    [ 759.        ,    0.0000322...],
                    [ 760.        ,    0.0000301...],
                    [ 761.        ,    0.0000281...],
                    [ 762.        ,    0.0000262...],
                    [ 763.        ,    0.0000244...],
                    [ 764.        ,    0.0000228...],
                    [ 765.        ,    0.0000213...],
                    [ 766.        ,    0.0000198...],
                    [ 767.        ,    0.0000185...],
                    [ 768.        ,    0.0000173...],
                    [ 769.        ,    0.0000161...],
                    [ 770.        ,    0.0000150...],
                    [ 771.        ,    0.0000140...],
                    [ 772.        ,    0.0000131...],
                    [ 773.        ,    0.0000122...],
                    [ 774.        ,    0.0000114...],
                    [ 775.        ,    0.0000106...],
                    [ 776.        ,    0.0000099...],
                    [ 777.        ,    0.0000092...],
                    [ 778.        ,    0.0000086...],
                    [ 779.        ,    0.0000080...],
                    [ 780.        ,    0.0000075...]],
                   interpolator=SpragueInterpolator,
                   interpolator_args={},
                   extrapolator=Extrapolator,
                   extrapolator_args={...})
```

Dataset

`colour`

LEFS	Aggregated luminous efficiency functions.
PHOTOPIC_LEFS	Photopic luminous efficiency functions.
SCOTOPIC_LEFS	Scotopic luminous efficiency functions.

colour.LEFS

`colour.`**`LEFS = CaseInsensitiveMapping({'CIE 1924 Photopic Standard Observer': ..., 'Judd Modified CIE 195`**

Aggregated luminous efficiency functions.

References

[CVRc], [CVRe], [Wik05e]

LEFS [CaseInsensitiveMapping] **{'CIE 1924 Photopic Standard Observer', 'Judd Modified CIE 1951 Photopic Standard Observer', 'Judd-Vos Modified CIE 1978 Photopic Standard Observer', 'CIE 1964 Photopic 10 Degree Standard Observer', 'CIE 2008 2 Degree Physiologically Relevant LEF', 'CIE 2008 10 Degree Physiologically Relevant LEF', 'CIE 1951 Scotopic Standard Observer'}**

colour.PHOTOPIC_LEFS

colour.**PHOTOPIC_LEFS = CaseInsensitiveMapping({'CIE 1924 Photopic Standard Observer': ..., 'Judd Modifie**
Photopic luminous efficiency functions.

References

[CVRc], [CVRe]

PHOTOPIC_LEFS [CaseInsensitiveMapping] **{'CIE 1924 Photopic Standard Observer', 'Judd Modified CIE 1951 Photopic Standard Observer', 'Judd-Vos Modified CIE 1978 Photopic Standard Observer', 'CIE 1964 Photopic 10 Degree Standard Observer', 'CIE 2008 2 Degree Physiologically Relevant LEF', 'CIE 2008 10 Degree Physiologically Relevant LEF'}**

Aliases:

- 'cie_2_1924': 'CIE 1931 2 Degree Standard Observer'
- 'cie_10_1964': 'CIE 1964 Photopic 10 Degree Standard Observer'

colour.SCOTOPIC_LEFS

colour.**SCOTOPIC_LEFS = CaseInsensitiveMapping({'CIE 1951 Scotopic Standard Observer': ..., 'cie_1951': .**
Scotopic luminous efficiency functions.

References

[CVRe]

SCOTOPIC_LEFS [CaseInsensitiveMapping] **{'CIE 1951 Scotopic Standard Observer', }**

Aliases:

- 'cie_1951': 'CIE 1951 Scotopic Standard Observer'

Lightness Computation

colour

lightness(Y[, method])	Returns the *Lightness* L of given *luminance* Y using given method.
LIGHTNESS_METHODS	Supported *Lightness* computation methods.

colour.lightness

colour.**lightness**(*Y*, *method='CIE 1976'*, ***kwargs*)
Returns the *Lightness* L of given *luminance* Y using given method.

Parameters

- **Y** (numeric or array_like) – *luminance* Y.
- **method** (unicode, optional) – **{'CIE 1976', 'Glasser 1958', 'Wyszecki 1963', 'Fairchild 2010', 'Fairchild 2011'}**, Computation method.

Other Parameters

- **Y_n** (*numeric or array_like, optional*) – {colour.colorimetry.lightness_CIE1976()}, White reference *luminance* Y_n.

- **epsilon** (*numeric or array_like, optional*) – {colour.colorimetry.lightness_Fairchild2010(), colour.colorimetry.lightness_Fairchild2011()}, ϵ exponent.

Returns *Lightness L*.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 100]	[0, 1]

References

[CIET14804f], [FW10], [FC11], [GMRS58], [Wik07d], [Wys63], [WS00c]

Examples

```
>>> lightness(12.19722535)
41.5278758...
>>> lightness(12.19722535, Y_n=100)
41.5278758...
>>> lightness(12.19722535, Y_n=95)
42.5199307...
>>> lightness(12.19722535, method='Glasser 1958')
39.8351264...
>>> lightness(12.19722535, method='Wyszecki 1963')
40.5475745...
>>> lightness(12.19722535, epsilon=0.710, method='Fairchild 2011')
...
29.8295108...
```

colour.LIGHTNESS_METHODS

colour.**LIGHTNESS_METHODS = CaseInsensitiveMapping({'Glasser 1958': ..., 'Wyszecki 1963': ..., 'CIE 1976'**

Supported *Lightness* computation methods.

References

[CIET14804f], [FW10], [FC11], [GMRS58], [Wys63], [WS00c]

LIGHTNESS_METHODS [CaseInsensitiveMapping] **{'Glasser 1958', 'Wyszecki 1963', 'CIE 1976', 'Fairchild 2010', 'Fairchild 2011'}**

Aliases:

- 'Lstar1976': 'CIE 1976'

Glasser, Mckinney, Reilly and Schnelle (1958)

colour.colorimetry

lightness_Glasser1958(Y)	Returns the *Lightness* L of given *luminance* Y using *Glasser et al.(1958)* method..

colour.colorimetry.lightness_Glasser1958

colour.colorimetry.**lightness_Glasser1958**(Y)

Returns the *Lightness* L of given *luminance* Y using *Glasser et al. (1958)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Lightness* L.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 100]	[0, 1]

References

[GMRS58]

Examples

```
>>> lightness_Glasser1958(12.19722535)
39.8351264...
```

Wyszecki (1963)

colour.colorimetry

lightness_Wyszecki1963(Y)	Returns the *Lightness* W of given *luminance* Y using *Wyszecki (1963)* method.

colour.colorimetry.lightness_Wyszecki1963

colour.colorimetry.**lightness_Wyszecki1963**(*Y*)

Returns the *Lightness* W of given *luminance* Y using *Wyszecki (1963)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Lightness* W.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
W	[0, 100]	[0, 1]

References

[Wys63]

Examples

```
>>> lightness_Wyszecki1963(12.19722535)
40.5475745...
```

CIE 1976

colour.colorimetry

lightness_CIE1976(Y[, Y_n])	Returns the *Lightness* L^* of given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.
intermediate_lightness_function_CIE1976(Y[, Y_n])	Returns the intermediate value $f(Y/Yn)$ in the *Lightness* L^* computation for given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

colour.colorimetry.lightness_CIE1976

colour.colorimetry.**lightness_CIE1976**(*Y*, *Y_n=100*)

Returns the *Lightness* L^* of given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **Y** (numeric or array_like) – *luminance* Y.
- **Y_n** (numeric or array_like, optional) – White reference *luminance* Y_n.

Returns *Lightness* L^*.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
L_star	[0, 100]	[0, 1]

References

[CIET14804f], [WS00c]

Examples

```
>>> lightness_CIE1976(12.19722535)
41.5278758...
```

colour.colorimetry.intermediate_lightness_function_CIE1976

colour.colorimetry.**intermediate_lightness_function_CIE1976**(*Y*, *Y_n=100*)

Returns the intermediate value $f(Y/Yn)$ in the *Lightness* L^* computation for given *luminance* Y using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **Y** (numeric or array_like) – *luminance* Y.
- **Y_n** (numeric or array_like, optional) – White reference *luminance* Y_n.

Returns Intermediate value $f(Y/Yn)$.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 100]

Range	Scale - Reference	Scale - 1
f_Y_Y_n	[0, 1]	[0, 1]

References

[CIET14804f], [WS00c]

Examples

```
>>> intermediate_lightness_function_CIE1976(12.19722535)
...
0.4959299...
>>> intermediate_lightness_function_CIE1976(12.19722535, 95)
...
0.5044821...
```

Fairchild and Wyble (2010)

`colour.colorimetry`

`lightness_Fairchild2010`(Y[, epsilon])	Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* kinetics.

colour.colorimetry.lightness_Fairchild2010

`colour.colorimetry.`**`lightness_Fairchild2010`**(*Y*, *epsilon=1.836*)

Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* kinetics.

Parameters

- **Y** (`array_like`) – *luminance* Y.
- **epsilon** (`numeric or array_like, optional`) – ϵ exponent.

Returns *Lightness* L_{hdr}.

Return type array_like

Warning: The input domain of that definition is non standard!

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

References

[FW10]

Examples

```
>>> lightness_Fairchild2010(12.19722535 / 100)
31.9963902...
```

Fairchild and Chen (2011)

colour.colorimetry

lightness_Fairchild2011(Y[, epsilon, method])	Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method according to *Michealis-Menten* kinetics.

colour.colorimetry.lightness_Fairchild2011

colour.colorimetry.**lightness_Fairchild2011**(*Y*, *epsilon=0.474*, *method='hdr-CIELAB'*)

Computes *Lightness* L_{hdr} of given *luminance* Y using *Fairchild and Chen (2011)* method according to *Michealis-Menten* kinetics.

Parameters

- **Y** (array_like) – *luminance* Y.
- **epsilon** (numeric or array_like, optional) – ϵ exponent.
- **method** (unicode, optional) – **{'hdr-CIELAB', 'hdr-IPT'}**, *Lightness* L_{hdr} computation method.

Returns *Lightness* L_{hdr}.

Return type array_like

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

References

[FC11]

Examples

```
>>> lightness_Fairchild2011(12.19722535 / 100)
51.8529584...
>>> lightness_Fairchild2011(12.19722535 / 100, method='hdr-IPT')
...
51.6431084...
```

Luminance Computation

colour

luminance(LV[, method])	Returns the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.
LUMINANCE_METHODS	Supported *luminance* computation methods.

colour.luminance

colour.**luminance**(*LV*, *method='CIE 1976'*, ***kwargs*)

Returns the *luminance* Y of given *Lightness* L^* or given *Munsell* value V.

Parameters

- **LV** (numeric or array_like) – *Lightness* L^* or *Munsell* value V.
- **method** (unicode, optional) – **{'CIE 1976', 'Newhall 1943', 'ASTM D1535', 'Fairchild 2010', 'Fairchild 2011'}**, Computation method.

Other Parameters

- **Y_n** (*numeric or array_like, optional*) – {colour.colorimetry.luminance_CIE1976()}, White reference *luminance* Y_n.
- **epsilon** (*numeric or array_like, optional*) – {colour.colorimetry.lightness_Fairchild2010(), colour.colorimetry.lightness_Fairchild2011()}, ϵ exponent.

Returns *luminance* Y.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
LV	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[ASTMInternational08], [CIET14804f], [FW10], [FC11], [NNJ43], [Wik01c], [WS00c]

Examples

```
>>> luminance(41.527875844653451)
12.1972253...
>>> luminance(41.527875844653451, Y_n=100)
12.1972253...
>>> luminance(42.51993072812094, Y_n=95)
12.1972253...
>>> luminance(4.08244375 * 10, method='Newhall 1943')
...
12.5500788...
>>> luminance(4.08244375 * 10, method='ASTM D1535')
...
12.2363426...
>>> luminance(29.829510892279330, epsilon=0.710, method='Fairchild 2011')
...
12.1972253...
```

colour.LUMINANCE_METHODS

colour.**LUMINANCE_METHODS = CaseInsensitiveMapping({'Newhall 1943': ..., 'ASTM D1535': ..., 'CIE 1976':**
Supported *luminance* computation methods.

References

[ASTMInternational08], [CIET14804f], [FW10], [FC11], [NNJ43], [WS00c]

LUMINANCE_METHODS [CaseInsensitiveMapping] **{'Newhall 1943', 'ASTM D1535', 'CIE 1976', 'Fairchild 2010'}**

Aliases:

- 'astm2008': 'ASTM D1535'
- 'cie1976': 'CIE 1976'

Newhall, Nickerson and Judd (1943)

colour.colorimetry

luminance_Newhall1943(V)	Returns the *luminance* R_Y of given *Munsell* value V using *Newhall et al.(1943)* method..

colour.colorimetry.luminance_Newhall1943

colour.colorimetry.**luminance_Newhall1943**(*V*)

Returns the *luminance* R_Y of given *Munsell* value V using *Newhall et al. (1943)* method.

Parameters **V** (numeric or array_like) – *Munsell* value V.

Returns *luminance* R_Y.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

Range	Scale - Reference	Scale - 1
R_Y	[0, 100]	[0, 1]

References

[NNJ43]

Examples

```
>>> luminance_Newhall1943(4.08244375)
12.5500788...
```

CIE 1976

colour.colorimetry

luminance_CIE1976(L_star[, Y_n])	Returns the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.
intermediate_luminance_function_CIE1976(f_Y_Yn)	Returns the *luminance* Y in the *luminance* Y computation for given intermediate value $f(Y/Yn)$ using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

colour.colorimetry.luminance_CIE1976

colour.colorimetry.**luminance_CIE1976**(*L_star*, *Y_n=100*)

Returns the *luminance* Y of given *Lightness* L^* with given reference white *luminance* Y_n.

Parameters

- **L_star** (numeric or array_like) – *Lightness* L^*
- **Y_n** (numeric or array_like) – White reference *luminance* Y_n.

Returns *luminance* Y.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
L_star	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[CIET14804f], [WS00c]

Examples

```
>>> luminance_CIE1976(41.527875844653451)
12.1972253...
>>> luminance_CIE1976(41.527875844653451, 95)
11.5873640...
```

colour.colorimetry.intermediate_luminance_function_CIE1976

colour.colorimetry.**intermediate_luminance_function_CIE1976**(*f_Y_Y_n*, *Y_n=100*)

Returns the *luminance* Y in the *luminance* Y computation for given intermediate value $f(Y/Yn)$ using given reference white *luminance* Y_n as per *CIE 1976* recommendation.

Parameters

- **f_Y_Y_n** (numeric or array_like) – Intermediate value $f(Y/Yn)$.
- **Y_n** (numeric or array_like) – White reference *luminance* Y_n.

Returns *luminance* Y.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
f_Y_Y_n	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 100]

References

[CIET14804f], [WS00c]

Examples

```
>>> intermediate_luminance_function_CIE1976(0.495929964178047)
...
12.1972253...
>>> intermediate_luminance_function_CIE1976(0.504482161449319, 95)
...
12.1972253...
```

ASTM D1535-08e1

colour.colorimetry

luminance_ASTMD1535(V)	Returns the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

colour.colorimetry.luminance_ASTMD1535

colour.colorimetry.**luminance_ASTMD1535**(V)

Returns the *luminance* Y of given *Munsell* value V using *ASTM D1535-08e1* method.

Parameters **V** (numeric or array_like) – *Munsell* value V.

Returns *luminance* Y.

Return type numeric or array_like

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

References

[ASTMInternational08]

Examples

```
>>> luminance_ASTMD1535(4.08244375)
12.2363426...
```

Fairchild and Wyble (2010)

colour.colorimetry

luminance_Fairchild2010(L_hdr[, epsilon])	Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* kinetics.

colour.colorimetry.luminance_Fairchild2010

colour.colorimetry.**luminance_Fairchild2010**(*L_hdr*, *epsilon=1.836*)

Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Wyble (2010)* method according to *Michealis-Menten* kinetics.

Parameters

- **L_hdr** (array_like) – *Lightness* L_{hdr}.
- **epsilon** (numeric or array_like, optional) – ϵ exponent.

Returns *luminance* Y.

Return type array_like

Warning: The output range of that definition is non standard!

Notes

Domain	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

References

[FW10]

Examples

```
>>> luminance_Fairchild2010(31.996390226262736, 1.836)
...
0.1219722...
```

Fairchild and Chen (2011)

colour.colorimetry

luminance_Fairchild2011(L_hdr[, epsilon, method])	Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method according to *Michealis-Menten* kinetics.

colour.colorimetry.luminance_Fairchild2011

colour.colorimetry.**luminance_Fairchild2011**(*L_hdr*, *epsilon=0.474*, *method='hdr-CIELAB'*)

Computes *luminance* Y of given *Lightness* L_{hdr} using *Fairchild and Chen (2011)* method according to *Michealis-Menten* kinetics.

Parameters

- **L_hdr** (array_like) – *Lightness* L_{hdr}.
- **epsilon** (numeric or array_like, optional) – ϵ exponent.
- **method** (unicode, optional) – **{'hdr-CIELAB', 'hdr-IPT'}**, *Lightness* L_{hdr} computation method.

Returns *luminance* Y.

Return type array_like

Notes

Domain	Scale - Reference	Scale - 1
L_hdr	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Y	[0, 1]	[0, 1]

References

[FC11]

Examples

```
>>> luminance_Fairchild2011(51.852958445912506)
0.1219722...
>>> luminance_Fairchild2011(51.643108411718522, method='hdr-IPT')
...
0.1219722...
```

Whiteness Computation

colour

whiteness(XYZ, XYZ_0[, method])	Returns the *whiteness* W using given method.
WHITENESS_METHODS	Supported *whiteness* computation methods.

colour.whiteness

colour.**whiteness**(*XYZ*, *XYZ_0*, *method='CIE 2004'*, ***kwargs*)

Returns the *whiteness* W using given method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white.
- **method** (unicode, optional) – **{'CIE 2004', 'Berger 1959', 'Taube 1960', 'Stensby 1968', 'ASTM E313', 'Ganz 1979'}**, Computation method.

Other Parameters observer (*unicode, optional*) – {colour.colorimetry.whiteness_CIE2004()}, **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**, *CIE Standard Observer* used for computations, *tint* T or T_{10} value is dependent on viewing field angular subtense.

Returns *whiteness* W.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
W	[0, 100]	[0, 1]

References

[CIET14804i], [WS00k], [XRP12], [Wik04c]

Examples

```
>>> import numpy as np
>>> from colour.models import xyY_to_XYZ
>>> XYZ = xyY_to_XYZ(np.array([0.3167, 0.3334, 100]))
>>> XYZ_0 = xyY_to_XYZ(np.array([0.3139, 0.3311, 100]))
>>> whiteness(XYZ, XYZ_0)
array([ 93.85...,  -1.305...])
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness(XYZ, XYZ_0, method='Taube 1960')
91.4071738...
```

colour.WHITENESS_METHODS

colour.**WHITENESS_METHODS = CaseInsensitiveMapping({'Berger 1959': ..., 'Taube 1960': ..., 'Stensby 1968'**
Supported *whiteness* computation methods.

References

[CIET14804i], [XRP12]

WHITENESS_METHODS [CaseInsensitiveMapping] **{'CIE 2004', 'Berger 1959', 'Taube 1960', 'Stensby 1968', 'ASTM E313', 'Ganz 1979', 'CIE 2004'}**

Aliases:

- 'cie2004': 'CIE 2004'

Berger (1959)

colour.colorimetry

whiteness_Berger1959(XYZ, XYZ_0)	Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Berger (1959)* method.

colour.colorimetry.whiteness_Berger1959

colour.colorimetry.**whiteness_Berger1959**(*XYZ*, *XYZ_0*)
Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Berger (1959)* method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white.

Returns *Whiteness* WI.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness* WI values larger than 33.33 indicate a bluish white and values smaller than 33.33 indicate a yellowish white.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness_Berger1959(XYZ, XYZ_0)
30.3638017...
```

Taube (1960)

colour.colorimetry

whiteness_Taube1960(XYZ, XYZ_0)	Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Taube (1960)* method.

colour.colorimetry.whiteness_Taube1960

colour.colorimetry.**whiteness_Taube1960**(*XYZ*, *XYZ_0*)

Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *Taube (1960)* method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.
- **XYZ_0** (array_like) – *CIE XYZ* tristimulus values of reference white.

Returns *Whiteness* WI.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_0	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness* WI values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> XYZ_0 = np.array([94.80966767, 100.00000000, 107.30513595])
>>> whiteness_Taube1960(XYZ, XYZ_0)
91.4071738...
```

Stensby (1968)

colour.colorimetry

whiteness_Stensby1968(Lab)	Returns the *whiteness* index WI of given sample *CIE L*a*b** colourspace array using *Stensby (1968)* method.

colour.colorimetry.whiteness_Stensby1968

colour.colorimetry.**whiteness_Stensby1968**(*Lab*)

Returns the *whiteness* index WI of given sample *CIE L*a*b** colourspace array using *Stensby (1968)* method.

Parameters **Lab** (array_like) – *CIE L*a*b** colourspace array of sample.

Returns *Whiteness* WI.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

- *Whiteness* WI values larger than 100 indicate a bluish white and values smaller than 100 indicate a yellowish white.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> Lab = np.array([100.00000000, -2.46875131, -16.72486654])
>>> whiteness_Stensby1968(Lab)
142.7683456...
```

ASTM E313

colour.colorimetry

whiteness_ASTME313(XYZ)	Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

colour.colorimetry.whiteness_ASTME313

colour.colorimetry.**whiteness_ASTME313**(*XYZ*)

Returns the *whiteness* index WI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.

Returns *Whiteness* WI.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> whiteness_ASTME313(XYZ)
55.7400000...
```

Ganz and Griesser (1979)

colour.colorimetry

whiteness_Ganz1979(xy, Y)	Returns the *whiteness* index W and *tint* T of given sample *CIE xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

colour.colorimetry.whiteness_Ganz1979

colour.colorimetry.**whiteness_Ganz1979**(*xy*, *Y*)

Returns the *whiteness* index W and *tint* T of given sample *CIE xy* chromaticity coordinates using *Ganz and Griesser (1979)* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates *CIE xy* of sample.
- **Y** (numeric or array_like) – Tristimulus Y value of sample.

Returns *Whiteness* W and *tint* T.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WT	[0, 100]	[0, 1]

- The formula coefficients are valid for *CIE Standard Illuminant D Series D65* and *CIE 1964 10 Degree Standard Observer*.
- Positive output *tint* T values indicate a greener tint while negative values indicate a redder tint.
- Whiteness differences of less than 5 Ganz units appear to be indistinguishable to the human eye.
- Tint differences of less than 0.5 Ganz units appear to be indistinguishable to the human eye.

References

[XRP12]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> whiteness_Ganz1979(xy, 100)
array([ 85.6003766...,   0.6789003...])
```

CIE 2004

colour.colorimetry

whiteness_CIE2004(xy, Y, xy_n[, observer])	Returns the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample *CIE xy* chromaticity coordinates using *CIE 2004* method.

colour.colorimetry.whiteness_CIE2004

colour.colorimetry.**whiteness_CIE2004**(*xy*, *Y*, *xy_n*, *observer='CIE 1931 2 Degree Standard Observer'*)

Returns the *whiteness* W or W_{10} and *tint* T or T_{10} of given sample *CIE xy* chromaticity coordinates using *CIE 2004* method.

Parameters

- **xy** (array_like) – Chromaticity coordinates *CIE xy* of sample.
- **Y** (numeric or array_like) – Tristimulus Y value of sample.
- **xy_n** (array_like) – Chromaticity coordinates *xy_n* of perfect diffuser.
- **observer** (unicode, optional) – **{'CIE 1931 2 Degree Standard Observer', 'CIE 1964 10 Degree Standard Observer'}**, *CIE Standard Observer* used for

computations, *tint* T or T_{10} value is dependent on viewing field angular subtense.

Returns *Whiteness* W or W_{10} and *tint* T or T_{10} of given sample.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
WT	[0, 100]	[0, 1]

- This method may be used only for samples whose values of W or W_{10} lie within the following limits: greater than 40 and less than 5Y - 280, or 5Y10 - 280.
- This method may be used only for samples whose values of T or T_{10} lie within the following limits: greater than -4 and less than +2.
- Output *whiteness* W or W_{10} values larger than 100 indicate a bluish white while values smaller than 100 indicate a yellowish white.
- Positive output *tint* T or T_{10} values indicate a greener tint while negative values indicate a redder tint.

References

[CIET14804i]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.3167, 0.3334])
>>> xy_n = np.array([0.3139, 0.3311])
>>> whiteness_CIE2004(xy, 100, xy_n)
array([ 93.85...,  -1.305...])
```

Yellowness Computation

colour

yellowness(XYZ[, method])	Returns the *yellowness* W using given method.
YELLOWNESS_METHODS	Supported *yellowness* computation methods.

colour.yellowness

colour.**yellowness**(*XYZ*, *method='ASTM E313'*)

Returns the *yellowness* W using given method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.
- **method** (unicode, optional) – **{'ASTM E313', 'ASTM D1925'}**, Computation method.

Returns *yellowness* Y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness(XYZ)
11.0650000...
>>> method = 'ASTM D1925'
>>> yellowness(XYZ, method=method)
10.2999999...
```

colour.YELLOWNESS_METHODS

colour.**YELLOWNESS_METHODS = CaseInsensitiveMapping({'ASTM D1925': ..., 'ASTM E313': ...})**

Supported *yellowness* computation methods.

References

[XRP12]

YELLOWNESS_METHODS [CaseInsensitiveMapping] **{'ASTM E313', 'ASTM D1925'}**

ASTM D1925

colour.colorimetry

yellowness_ASTMD1925(XYZ)	Returns the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM D1925* method.

colour.colorimetry.yellowness_ASTMD1925

colour.colorimetry.**yellowness_ASTMD1925**(*XYZ*)

Returns the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM D1925* method.

ASTM D1925 has been specifically developed for the definition of the Yellowness of homogeneous, non-fluorescent, almost neutral-transparent, white-scattering or opaque plastics as they will be reviewed under daylight condition. It can be other materials as well, as long as they fit into this description.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.

Returns *Whiteness* YI.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTMD1925(XYZ)
10.2999999...
```

ASTM E313

colour.colorimetry

yellowness_ASTME313(XYZ)	Returns the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

colour.colorimetry.yellowness_ASTME313

colour.colorimetry.**yellowness_ASTME313**(*XYZ*)

Returns the *yellowness* index YI of given sample *CIE XYZ* tristimulus values using *ASTM E313* method.

ASTM E313 has successfully been used for a variety of white or near white materials. This includes coatings, Plastics, Textiles.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values of sample.

Returns *Whiteness* YI.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
YI	[0, 100]	[0, 1]

References

[XRP12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([95.00000000, 100.00000000, 105.00000000])
>>> yellowness_ASTME313(XYZ)
11.0650000...
```

Constants

- *CIE*
- *CODATA*
- *Common*

CIE

colour.constants

K_M	Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.
KP_M	Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

colour.constants.K_M

colour.constants.**K_M = 683.0**

Rounded maximum photopic luminous efficiency K_m value in $lm \cdot W^{-1}$.

K_M : numeric

Notes

- To be adequate for all practical applications the K_m value has been rounded from the original 683.002 value.

References

[WS00g]

colour.constants.KP_M

colour.constants.**KP_M = 1700.0**

Rounded maximum scotopic luminous efficiency K'_m value in $lm \cdot W^{-1}$.

KP_M : numeric

Notes

- To be adequate for all practical applications the K'_m value has been rounded from the original 1700.06 value.

References

[WS00g]

CODATA

colour.constants

AVOGADRO_CONSTANT	Avogadro constant.
BOLTZMANN_CONSTANT	Boltzmann constant.
LIGHT_SPEED	Speed of light in vacuum.
PLANCK_CONSTANT	Planck constant.

colour.constants.AVOGADRO_CONSTANT

colour.constants.**AVOGADRO_CONSTANT = 6.02214179e+23**
Avogadro constant.

AVOGADRO_CONSTANT : numeric

colour.constants.BOLTZMANN_CONSTANT

colour.constants.**BOLTZMANN_CONSTANT = 1.38065e-23**
Boltzmann constant.

BOLTZMANN_CONSTANT : numeric

colour.constants.LIGHT_SPEED

colour.constants.**LIGHT_SPEED = 299792458.0**
Speed of light in vacuum.

LIGHT_SPEED : numeric

colour.constants.PLANCK_CONSTANT

colour.constants.**PLANCK_CONSTANT = 6.62607e-34**
Planck constant.

PLANCK_CONSTANT : numeric

Common

colour.constants

DEFAULT_FLOAT_DTYPE	alias of numpy.float64
DEFAULT_INT_DTYPE	alias of numpy.int64
EPSILON	
FLOATING_POINT_NUMBER_PATTERN	str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str
INTEGER_THRESHOLD	Integer threshold value.

colour.constants.DEFAULT_FLOAT_DTYPE

colour.constants.**DEFAULT_FLOAT_DTYPE**
alias of numpy.float64

colour.constants.DEFAULT_INT_DTYPE

colour.constants.**DEFAULT_INT_DTYPE**
alias of numpy.int64

colour.constants.EPSILON

colour.constants.**EPSILON = 2.2204460492503131e-16**

colour.constants.FLOATING_POINT_NUMBER_PATTERN

colour.constants.**FLOATING_POINT_NUMBER_PATTERN = '[0-9]*\\.?[0-9]+([eE][-+]?[0-9]+)?'**
str(object='') -> str str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or errors is specified, then the object must expose a data buffer that will be decoded using the given encoding and error handler. Otherwise, returns the result of object.__str__() (if defined) or repr(object). encoding defaults to sys.getdefaultencoding(). errors defaults to 'strict'.

colour.constants.INTEGER_THRESHOLD

colour.constants.**INTEGER_THRESHOLD = 0.001**
Integer threshold value.

INTEGER_THRESHOLD : numeric

Contrast Sensitivity

- *Contrast Sensitivity*
- *Barten (1999) Contrast Sensitivity Function*

Contrast Sensitivity

colour

contrast_sensitivity_function([method])	Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.
CONTRAST_SENSITIVITY_METHODS	Supported contrast sensitivity methods.

colour.contrast_sensitivity_function

colour.**contrast_sensitivity_function**(*method='Barten 1999'*, ***kwargs*)

Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by given method.

Parameters **method** (unicode, optional) – **{'Barten 1999'}**, Computation method.

Other Parameters

- **E** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Retinal illuminance E in Trolands.
- **N_max** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Maximum number of cycles N_{max} over which the eye can integrate the information.
- **T** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Integration time T in seconds of the eye.
- **X_0** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Angular size X_0 in degrees of the object in the x direction.
- **Y_0** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Angular size Y_0 in degrees of the object in the y direction.
- **X_max** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **Y_max** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Maximum angular size Y_{max} in degrees of the integration area in the y direction.
- **k** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Signal-to-noise (SNR) ratio k.
- **n** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Quantum efficiency of the eye n.
- **p** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Photon conversion factor p in $photons \div seconds \div degrees^2 \div Trolands$ that depends on the light source.
- **phi_0** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Spectral density ϕ_0 in $seconds degrees^2$ of the neural noise.
- **sigma** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.
- **u** (*numeric*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Spatial frequency u, the cycles per degree.
- **u_0** (*numeric or array_like, optional*) – {colour.contrast.contrast_sensitivity_function_Barten1999()}, Spatial frequency u_0 in $cycles \div degrees$ above which the lateral inhibition ceases.

Returns Contrast sensitivity S.

Return type ndarray

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> contrast_sensitivity_function(u=4)
360.8691122...
>>> contrast_sensitivity_function('Barten 1999', u=4)
360.8691122...
```

colour.CONTRAST_SENSITIVITY_METHODS

colour.**CONTRAST_SENSITIVITY_METHODS = CaseInsensitiveMapping({'Barten 1999': ...})**

Supported contrast sensitivity methods.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

CONTRAST_SENSITIVITY_METHODS [CaseInsensitiveMapping] **{'Barten 1999'}**

Barten (1999) Contrast Sensitivity Function

colour.contrast

contrast_sensitivity_function_Barten1999(u)	Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by *Barten (1999)*.

colour.contrast.contrast_sensitivity_function_Barten1999

colour.contrast.**contrast_sensitivity_function_Barten1999**(*u*, *sigma=0.008791157173230633*, *k=3.0*, *T=0.1*, *X_0=60*, *Y_0=None*, *X_max=12*, *Y_max=None*, *N_max=15*, *n=0.03*, *p=1227400.0*, *E=66.082316060529919*, *phi_0=3.0000000000000004e-08*, *u_0=7*)

Returns the contrast sensitivity S of the human eye according to the contrast sensitivity function (CSF) described by *Barten (1999)*.

Contrast sensitivity is defined as the inverse of the modulation threshold of a sinusoidal luminance pattern. The modulation threshold of this pattern is generally defined by 50% probability of detection. The contrast sensitivity function or CSF gives the contrast sensitivity as a function of spatial frequency. In the CSF, the spatial frequency is expressed in angular units with respect to the eye. It reaches a maximum between 1 and 10 cycles per degree with a fall off at higher and lower spatial frequencies.

Parameters

- **u** (numeric) – Spatial frequency u, the cycles per degree.
- **sigma** (numeric or array_like, optional) – Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.
- **k** (numeric or array_like, optional) – Signal-to-noise (SNR) ratio k.
- **T** (numeric or array_like, optional) – Integration time T in seconds of the eye.
- **X_0** (numeric or array_like, optional) – Angular size X_0 in degrees of the object in the x direction.
- **Y_0** (numeric or array_like, optional) – Angular size Y_0 in degrees of the object in the y direction.
- **X_max** (numeric or array_like, optional) – Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **Y_max** (numeric or array_like, optional) – Maximum angular size Y_{max} in degrees of the integration area in the y direction.
- **N_max** (numeric or array_like, optional) – Maximum number of cycles N_{max} over which the eye can integrate the information.
- **n** (numeric or array_like, optional) – Quantum efficiency of the eye n.
- **p** (numeric or array_like, optional) – Photon conversion factor p in $photons \div seconds \div degrees^2 \div Trolands$ that depends on the light source.
- **E** (numeric or array_like, optional) – Retinal illuminance E in Trolands.
- **phi_0** (numeric or array_like, optional) – Spectral density ϕ_0 in $seconds degrees^2$ of the neural noise.
- **u_0** (numeric or array_like, optional) – Spatial frequency u_0 in $cycles \div degrees$ above which the lateral inhibition ceases.

Returns Contrast sensitivity S.

Return type ndarray

Warning: This definition expects σ_0 and C_{ab} used in the computation of σ to be given in degrees and $degrees \div mm$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in $arcmin$ and $arcmin \div mm$ respectively, thus they need to be divided by 60.

Notes

- The formula holds for bilateral viewing and for equal dimensions of the object in x and y direction. For monocular vision, the contrast sensitivity is a factor $\sqrt{2}$ smaller.
- *Barten (1999)* CSF default values for the k, σ_0, C_{ab}, T, X_{max}, N_{max}, n, ϕ_0 and u_0 constants are valid for a standard observer with good vision and with an age between 20 and 30 years.
- The other constants have been filled using reference data from *Figure 31* in [InternationalTUnion15c] but must be adapted to the current use case.
- The product of u, the cycles per degree, and X_0, the number of degrees, gives the number of cycles P_c in a pattern. Therefore, X_0 can be made a variable dependent on u such as $X_0 = P_c/u$.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> contrast_sensitivity_function_Barten1999(4)
360.8691122...
```

Reproducing *Figure 31* in [InternationalTUnion15c] illustrating the minimum detectable contrast according to *Barten (1999)* model with the assumed conditions for UHDTV applications. The minimum detectable contrast MDC is then defined as follows:

```
:math:`MDC = 1 / CSF * 2 * (1 / 1.27)`
```

where 2 is used for the conversion from modulation to contrast and $1/1.27$ is used for the conversion from sinusoidal to rectangular waves.

```
>>> from scipy.optimize import fmin
>>> settings_BT2246 = {
...     'k': 3.0,
...     'T': 0.1,
...     'X_max': 12,
...     'N_max': 15,
...     'n': 0.03,
...     'p': 1.2274 * 10 ** 6,
...     'phi_0': 3 * 10 ** -8,
...     'u_0': 7,
... }
>>>
>>> def maximise_spatial_frequency(L):
...     maximised_spatial_frequency = []
...     for L_v in L:
...         X_0 = 60
...         d = pupil_diameter_Barten1999(L_v, X_0)
...         sigma = sigma_Barten1999(0.5 / 60, 0.08 / 60, d)
...         E = retinal_illuminance_Barten1999(L_v, d, True)
...         maximised_spatial_frequency.append(
...             fmin(lambda x: (
...                     -contrast_sensitivity_function_Barten1999(
...                         u=x,
...                         sigma=sigma,
...                         X_0=X_0,
...                         E=E,
...                         **settings_BT2246)
...                 ), 0, disp=False)[0])
...         return as_float(np.array(maximised_spatial_frequency))
>>>
>>> L = np.logspace(np.log10(0.01), np.log10(100), 10)
>>> X_0 = Y_0 = 60
>>> d = pupil_diameter_Barten1999(L, X_0, Y_0)
>>> sigma = sigma_Barten1999(0.5 / 60, 0.08 / 60, d)
>>> E = retinal_illuminance_Barten1999(L, d)
>>> u = maximise_spatial_frequency(L)
>>> (1 / contrast_sensitivity_function_Barten1999(
...     u=u, sigma=sigma, E=E, X_0=X_0, Y_0=Y_0, **settings_BT2246)
...  * 2 * (1/ 1.27))
...
array([ 0.0207396...,  0.0134885...,  0.0096063...,  0.0077299...,  0.0068983...,
        0.0065057...,  0.0062712...,  0.0061198...,  0.0060365...,  0.0059984...])
```

Ancillary Objects

colour.contrast

optical_MTF_Barten1999(u[, sigma])	Returns the optical modulation transfer function (MTF) M_{opt} of the eye using *Barten (1999)* method.
pupil_diameter_Barten1999(L[, X_0, Y_0])	Returns the pupil diameter for given luminance and object or stimulus angular size using *Barten (1999)* method.
sigma_Barten1999([sigma_0, C_ab, d])	Returns the standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process using *Barten (1999)* method.
retinal_illuminance_Barten1999(L[, d, ...])	Returns the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using *Barten (1999)* method.
maximum_angular_size_Barten1999(u[, X_0, ...])	Returns the maximum angular size X of the object considered using *Barten (1999)* method.

colour.contrast.optical_MTF_Barten1999

colour.contrast.**optical_MTF_Barten1999**(*u*, *sigma=0.01*)

Returns the optical modulation transfer function (MTF) M_{opt} of the eye using *Barten (1999)* method.

Parameters

- **u** (numeric or array_like) – Spatial frequency u, the cycles per degree.
- **sigma** (numeric or array_like, optional) – Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.

Returns Optical modulation transfer function (MTF) M_{opt} of the eye.

Return type numeric or array_like

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> optical_MTF_Barten1999(4, 0.01)
0.9689107...
```

colour.contrast.pupil_diameter_Barten1999

colour.contrast.**pupil_diameter_Barten1999**(*L*, *X_0=60*, *Y_0=None*)

Returns the pupil diameter for given luminance and object or stimulus angular size using *Barten (1999)* method.

Parameters

- **L** (numeric or array_like) – Average luminance L in cd/m^2.
- **X_0** (numeric or array_like, optional) – Angular size of the object X_0 in degrees in the x direction.
- **Y_0** (numeric or array_like, optional) – Angular size of the object X_0 in degrees in the y direction.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> pupil_diameter_Barten1999(100, 60, 60)
2.0777571...
```

colour.contrast.sigma_Barten1999

colour.contrast.**sigma_Barten1999**(*sigma_0=0.008333333333333333*, *C_ab=0.0013333333333333333*, *d=2.1*)

Returns the standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process using *Barten (1999)* method.

The σ quantity depends on the pupil diameter d of the eye lens. For very small pupil diameters, σ increases inversely proportionally with pupil size because of diffraction, and for large pupil diameters, σ increases about linearly with pupil size because of chromatic aberration and others aberrations.

Parameters

- **sigma_0** (numeric or array_like, optional) – Constant σ_0 in degrees.
- **C_ab** (numeric or array_like, optional) – Spherical aberration of the eye C_{ab} in $degrees \div mm$.
- **d** (numeric or array_like, optional) – Pupil diameter d in millimeters.

Returns Standard deviation σ of the line-spread function resulting from the convolution of the different elements of the convolution process.

Return type ndarray

Warning: This definition expects σ_0 and C_{ab} to be given in degrees and $degrees \div mm$ respectively. However, in the literature, the values for σ_0 and C_{ab} are usually given in $arcmin$ and $arcmin \div mm$ respectively, thus they need to be divided by 60.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> sigma_Barten1999(0.5 / 60, 0.08 / 60, 2.1)
0.0087911...
```

colour.contrast.retinal_illuminance_Barten1999

colour.contrast.**retinal_illuminance_Barten1999**(*L*, *d=2.1*, *apply_stiles_crawford_effect_correction=True*)

Returns the retinal illuminance E in Trolands for given average luminance L and pupil diameter d using *Barten (1999)* method.

Parameters

- **L** (numeric or array_like) – Average luminance L in cd/m^2.
- **d** (numeric or array_like, optional) – Pupil diameter d in millimeters.
- **apply_stiles_crawford_effect_correction** (bool, optional) – Whether to apply the correction for *Stiles-Crawford* effect.

Returns Retinal illuminance E in Trolands.

Return type ndarray

Notes

- This definition is for use with photopic viewing conditions and thus corrects for the Stiles-Crawford effect by default, i.e. directional sensitivity of the cone cells with lower response of cone cells receiving light from the edge of the pupil.

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> retinal_illuminance_Barten1999(100, 2.1)
330.4115803...
>>> retinal_illuminance_Barten1999(100, 2.1, False)
346.3605900...
```

colour.contrast.maximum_angular_size_Barten1999

colour.contrast.**maximum_angular_size_Barten1999**(*u*, *X_0=60*, *X_max=12*, *N_max=15*)

Returns the maximum angular size X of the object considered using *Barten (1999)* method.

Parameters

- **u** (numeric) – Spatial frequency u, the cycles per degree.
- **X_0** (numeric or array_like, optional) – Angular size X_0 in degrees of the object in the x direction.
- **X_max** (numeric or array_like, optional) – Maximum angular size X_{max} in degrees of the integration area in the x direction.
- **N_max** (numeric or array_like, optional) – Maximum number of cycles N_{max} over which the eye can integrate the information.

Returns Maximum angular size X of the object considered.

Return type numeric or ndarray

References

[Bar99], [Bar03], [CKM+04], [InternationalTUnion15c],

Examples

```
>>> maximum_angular_size_Barten1999(4)
3.5729480...
```

Continuous Signal

- *Continuous Signal*

Continuous Signal

colour.continuous

AbstractContinuousFunction([name])	Defines the base class for abstract continuous function.
Signal([data, domain])	Defines the base class for continuous signal.
MultiSignals([data, domain, labels])	Defines the base class for multi-continuous signals, a container for multiple colour.continuous.Signal sub-class instances.

colour.continuous.AbstractContinuousFunction

class colour.continuous.**AbstractContinuousFunction**(*name=None*)

Bases: object

Defines the base class for abstract continuous function.

This is an ABCMeta abstract class that must be inherited by sub-classes.

The sub-classes are expected to implement the colour.continuous.AbstractContinuousFunction.function() method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable. A conventional implementation adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the colour.continuous.AbstractContinuousFunction.domain attribute corresponds with the function dependent and already known range stored in the colour.continuous.AbstractContinuousFunction.range attribute.

Parameters **name** (unicode, optional) – Continuous function name.

name

domain

range

interpolator

interpolator_args

extrapolator

extrapolator_args

function

__str__()

__repr__()

__hash__()

__getitem__()

__setitem__()

__contains__()

__len__()

__eq__()

__ne__()

__iadd__()

__add__()

__isub__()

__sub__()

__imul__()

__mul__()

__idiv__()

__div__()

__ipow__()

__pow__()

arithmetical_operation()

fill_nan()

domain_distance()

is_uniform()

copy()

abstract arithmetical_operation(*a*, *operation*, *in_place=False*)
Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place, must be reimplemented by sub-classes.

Parameters

- **a** (numeric or ndarray or AbstractContinuousFunction) – Operand.
- **operation** (object) – Operation to perform.
- **in_place** (bool, optional) – Operation happens in place.

Returns Abstract continuous function.

Return type *AbstractContinuousFunction*

copy()
Returns a copy of the sub-class instance.

Returns Abstract continuous function copy.

Return type *AbstractContinuousFunction*

abstract property domain
Getter and setter property for the abstract continuous function independent domain x variable, must be reimplemented by sub-classes.

Parameters **value** (array_like) – Value to set the abstract continuous function independent domain x variable with.

Returns Abstract continuous function independent domain x variable.

Return type ndarray

domain_distance(*a*)
Returns the euclidean distance between given array and independent domain x closest element.

Parameters **a** (numeric or array_like) – a variable to compute the euclidean distance with independent domain x variable.

Returns Euclidean distance between independent domain x variable and given a variable.

Return type numeric or array_like

abstract property extrapolator
Getter and setter property for the abstract continuous function extrapolator type, must be reimplemented by sub-classes.

Parameters **value** (type) – Value to set the abstract continuous function extrapolator type with.

Returns Abstract continuous function extrapolator type.

Return type type

abstract property extrapolator_args
Getter and setter property for the abstract continuous function extrapolator instantiation time arguments, must be reimplemented by sub-classes.

Parameters **value** (dict) – Value to set the abstract continuous function extrapolator instantiation time arguments to.

Returns Abstract continuous function extrapolator instantiation time arguments.

Return type dict

abstract fill_nan(*method='Interpolation'*, *default=0*)
Fill NaNs in independent domain x variable and corresponding range y variable using given method, must be reimplemented by sub-classes.

Parameters

- **method** (unicode, optional) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with `default`.
- **default** (numeric, optional) – Value to use with the *Constant* method.

Returns NaNs filled abstract continuous function.

Return type *AbstractContinuousFunction*

abstract property function
Getter and setter property for the abstract continuous function callable, must be reimplemented by sub-classes.

Parameters **value** (object) – Attribute value.

Returns Abstract continuous function callable.

Return type callable

Notes

- This property is read only.

abstract property interpolator
Getter and setter property for the abstract continuous function interpolator type, must be reimplemented by sub-classes.

Parameters **value** (type) – Value to set the abstract continuous function interpolator type with.

Returns Abstract continuous function interpolator type.

Return type type

abstract property interpolator_args
Getter and setter property for the abstract continuous function interpolator instantiation time arguments, must be reimplemented by sub-classes.

Parameters **value** (dict) – Value to set the abstract continuous function interpolator instantiation time arguments to.

Returns Abstract continuous function interpolator instantiation time arguments.

Return type dict

is_uniform()
Returns if independent domain x variable is uniform.

Returns Is independent domain x variable uniform.

Return type bool

property name
Getter and setter property for the abstract continuous function name.

Parameters **value** (unicode) – Value to set the abstract continuous function name with.

Returns Abstract continuous function name.

Return type unicode

abstract property range

Getter and setter property for the abstract continuous function corresponding range y variable, must be reimplemented by sub-classes.

Parameters **value** (array_like) – Value to set the abstract continuous function corresponding range y variable with.

Returns Abstract continuous function corresponding range y variable.

Return type ndarray

colour.continuous.Signal

class colour.continuous.**Signal**(*data=None*, *domain=None*, ***kwargs*)

Bases: colour.continuous.abstract.AbstractContinuousFunction

Defines the base class for continuous signal.

The class implements the Signal.function() method so that evaluating the function for any independent domain $x \in \mathbb{R}$ variable returns a corresponding range $y \in \mathbb{R}$ variable. It adopts an interpolating function encapsulated inside an extrapolating function. The resulting function independent domain, stored as discrete values in the colour.continuous.Signal.domain attribute corresponds with the function dependent and already known range stored in the colour.continuous.Signal.range attribute.

Parameters

- **data** (Series or Signal or array_like or dict_like, optional) – Data to be stored in the continuous signal.
- **domain** (array_like, optional) – Values to initialise the colour.continuous.Signal.domain attribute with. If both data and domain arguments are defined, the latter with be used to initialise the colour.continuous.Signal.domain attribute.

Other Parameters

- **name** (*unicode, optional*) – Continuous signal name.
- **dtype** (*type, optional*) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function.

dtype

domain

range

interpolator

interpolator_args

extrapolator

extrapolator_args

function

__str__()

__repr__()

__hash__()

__getitem__()

__setitem__()

__contains__()

__eq__()

__ne__()

arithmetical_operation()

signal_unpack_data()

fill_nan()

to_series()

Examples

Instantiation with implicit *domain*:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(Signal(range_))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

Instantiation with explicit *domain*:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(Signal(range_, domain))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Instantiation with a *dict*:

```
>>> print(Signal(dict(zip(domain, range_))))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
```

(continues on next page)

(continued from previous page)

```
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Instantiation with a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     print(Signal(
...         Series(dict(zip(domain, range_)))))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Retrieving domain *y* variable for arbitrary range *x* variable:

```
>>> x = 150
>>> range_ = np.sin(np.linspace(0, 1, 10))
>>> Signal(range_, domain)[x]
0.0359701...
>>> x = np.linspace(100, 1000, 3)
>>> Signal(range_, domain)[x]
array([  ...,   4.7669395...e-01,   8.4147098...e-01])
```

Using an alternative interpolating function:

```
>>> x = 150
>>> from colour.algebra import CubicSplineInterpolator
>>> Signal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
0.0555274...
>>> x = np.linspace(100, 1000, 3)
>>> Signal(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
array([ 0.        ,  0.4794253...,  0.8414709...])
```

arithmetical_operation(*a*, *operation*, *in_place=False*)

Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.

Parameters

- **a** (numeric or ndarray or Signal) – Operand.
- **operation** (object) – Operation to perform.
- **in_place** (bool, optional) – Operation happens in place.

Returns Continuous signal.

Return type *Signal*

Examples

Adding a single *numeric* variable:

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal_1 = Signal(range_)
>>> print(signal_1)
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> print(signal_1.arithmetical_operation(10, '+', True))
[[   0.   20.]
 [   1.   30.]
 [   2.   40.]
 [   3.   50.]
 [   4.   60.]
 [   5.   70.]
 [   6.   80.]
 [   7.   90.]
 [   8.  100.]
 [   9.  110.]]
```

Adding an *array_like* variable:

```
>>> a = np.linspace(10, 100, 10)
>>> print(signal_1.arithmetical_operation(a, '+', True))
[[   0.   30.]
 [   1.   50.]
 [   2.   70.]
 [   3.   90.]
 [   4.  110.]
 [   5.  130.]
 [   6.  150.]
 [   7.  170.]
 [   8.  190.]
 [   9.  210.]]
```

Adding a `colour.continuous.Signal` class:

```
>>> signal_2 = Signal(range_)
>>> print(signal_1.arithmetical_operation(signal_2, '+', True))
[[   0.   40.]
 [   1.   70.]
 [   2.  100.]
 [   3.  130.]
 [   4.  160.]
 [   5.  190.]
 [   6.  220.]
 [   7.  250.]
 [   8.  280.]
 [   9.  310.]]
```

property domain
Getter and setter property for the continuous signal independent domain x variable.

Parameters **value** (array_like) – Value to set the continuous signal independent domain x variable with.

Returns Continuous signal independent domain x variable.

Return type ndarray

property dtype
Getter and setter property for the continuous signal dtype.

Parameters **value** (type) – Value to set the continuous signal dtype with.

Returns Continuous signal dtype.

Return type type

property extrapolator
Getter and setter property for the continuous signal extrapolator type.

Parameters **value** (type) – Value to set the continuous signal extrapolator type with.

Returns Continuous signal extrapolator type.

Return type type

property extrapolator_args
Getter and setter property for the continuous signal extrapolator instantiation time arguments.

Parameters **value** (dict) – Value to set the continuous signal extrapolator instantiation time arguments to.

Returns Continuous signal extrapolator instantiation time arguments.

Return type dict

fill_nan(*method='Interpolation'*, *default=0*)
Fill NaNs in independent domain x variable and corresponding range y variable using given method.

Parameters

- **method** (unicode, optional) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (numeric, optional) – Value to use with the *Constant* method.

Returns NaNs filled continuous signal.

Return type *Signal*

Examples

```
>>> range_ = np.linspace(10, 100, 10)
>>> signal = Signal(range_)
>>> signal[3:7] = np.nan
>>> print(signal)
[[  0.   10.]
 [  1.   20.]
 [  2.   30.]
 [  3.   nan]
 [  4.   nan]
 [  5.   nan]
 [  6.   nan]
 [  7.   80.]
```

(continues on next page)

(continued from previous page)

```
 [   8.   90.]
 [   9.  100.]]
>>> print(signal.fill_nan())
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
>>> signal[3:7] = np.nan
>>> print(signal.fill_nan(method='Constant'))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.    0.]
 [   4.    0.]
 [   5.    0.]
 [   6.    0.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

property function

Getter and setter property for the continuous signal callable.

Parameters **value** (object) – Attribute value.

Returns Continuous signal callable.

Return type callable

Notes

- This property is read only.

property interpolator

Getter and setter property for the continuous signal interpolator type.

Parameters **value** (type) – Value to set the continuous signal interpolator type with.

Returns Continuous signal interpolator type.

Return type type

property interpolator_args

Getter and setter property for the continuous signal interpolator instantiation time arguments.

Parameters **value** (dict) – Value to set the continuous signal interpolator instantiation time arguments to.

Returns Continuous signal interpolator instantiation time arguments.

Return type dict

property range

Getter and setter property for the continuous signal corresponding range y variable.

Parameters **value** (array_like) – Value to set the continuous signal corresponding range y variable with.

Returns Continuous signal corresponding range y variable.

Return type ndarray

static signal_unpack_data(*data=None*, *domain=None*, *dtype=<class 'numpy.float64'>*)
Unpack given data for continuous signal instantiation.

Parameters

- **data** (Series or Signal or array_like or dict_like, optional) – Data to unpack for continuous signal instantiation.
- **domain** (array_like, optional) – Values to initialise the colour.continuous.Signal.domain attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.domain attribute.
- **dtype** (type, optional) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.

Returns Independent domain x variable and corresponding range y variable unpacked for continuous signal instantiation.

Return type tuple

Examples

Unpacking using implicit *domain*:

```
>>> range_ = np.linspace(10, 100, 10)
>>> domain, range_ = Signal.signal_unpack_data(range_)
>>> print(domain)
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using explicit *domain*:

```
>>> domain = np.arange(100, 1100, 100)
>>> domain, range = Signal.signal_unpack_data(range_, domain)
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a *dict*:

```
>>> domain, range_ = Signal.signal_unpack_data(
...     dict(zip(domain, range_)))
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     domain, range = Signal.signal_unpack_data(
...         Series(dict(zip(domain, range_))))
...
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

Unpacking using a colour.continuous.Signal class:

```
>>> domain, range_ = Signal.signal_unpack_data(
...     Signal(range_, domain))
>>> print(domain)
[  100.   200.   300.   400.   500.   600.   700.   800.   900.  1000.]
>>> print(range_)
[  10.   20.   30.   40.   50.   60.   70.   80.   90.  100.]
```

to_series()

Converts the continuous signal to a *Pandas* Series class instance.

Returns Continuous signal as a *Pandas* Series class instance.

Return type Series

Examples

```
>>> if is_pandas_installed():
...     range_ = np.linspace(10, 100, 10)
...     signal = Signal(range_)
...     print(signal.to_series())
0.0     10.0
1.0     20.0
2.0     30.0
3.0     40.0
4.0     50.0
5.0     60.0
6.0     70.0
7.0     80.0
8.0     90.0
9.0    100.0
Name: Signal (...), dtype: float64
```

colour.continuous.MultiSignals

class colour.continuous.**MultiSignals**(*data=None*, *domain=None*, *labels=None*, ***kwargs*)

Bases: colour.continuous.abstract.AbstractContinuousFunction

Defines the base class for multi-continuous signals, a container for multiple colour.continuous.Signal sub-class instances.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or array_like or dict_like, optional) – Data to be stored in the multi-continuous signals.
- **domain** (array_like, optional) – Values to initialise the multiple colour.continuous.Signal sub-class instances colour.continuous.Signal.domain attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.domain attribute.
- **labels** (array_like, optional) – Names to use for the colour.continuous.Signal sub-class instances.

Other Parameters

- **name** (*unicode, optional*) – multi-continuous signals name.
- **dtype** (*type, optional*) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the `colour.continuous.Signal` sub-class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.continuous.Signal` sub-class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.continuous.Signal` sub-class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.continuous.Signal` sub-class instances.
- **signal_type** (*type, optional*) – The `colour.continuous.Signal` sub-class type used for instances.

dtype

domain

range

interpolator

interpolator_args

extrapolator

extrapolator_args

function

signals

labels

signal_type

__str__()

__repr__()

__hash__()

__getitem__()

__setitem__()

__contains__()

__eq__()

__ne__()

arithmetical_operation()

multi_signals_unpack_data()

fill_nan()

to_dataframe()

Examples

Instantiation with implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> print(MultiSignals(range_))
[[   0.   10.]
 [   1.   20.]
 [   2.   30.]
 [   3.   40.]
 [   4.   50.]
 [   5.   60.]
 [   6.   70.]
 [   7.   80.]
 [   8.   90.]
 [   9.  100.]]
```

Instantiation with explicit *domain* and a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> print(MultiSignals(range_, domain))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Instantiation with multiple signals:

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> print(MultiSignals(range_, domain))
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
```

Instantiation with a *dict*:

```
>>> print(MultiSignals(dict(zip(domain, range_))))
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
```

Instantiation using a *Signal* sub-class:

```
>>> class NotSignal(Signal):
...     pass
```

```
>>> multi_signals = MultiSignals(range_, domain, signal_type=NotSignal)
>>> print(multi_signals)
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
 >>> type(multi_signals.signals[0])
 <class 'multi_signals.NotSignal'>
```

Instantiation with a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     print(MultiSignals(
...         Series(dict(zip(domain, np.linspace(10, 100, 10))))))
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Instantiation with a *Pandas Dataframe*:

```
>>> if is_pandas_installed():
...     from pandas import DataFrame
...     data = dict(zip(['a', 'b', 'c'], tsplit(range_)))
...     print(MultiSignals(
...         DataFrame(data, domain)))
[[  100.    10.    20.    30.]
 [  200.    20.    30.    40.]
 [  300.    30.    40.    50.]
 [  400.    40.    50.    60.]
 [  500.    50.    60.    70.]
 [  600.    60.    70.    80.]
 [  700.    70.    80.    90.]
 [  800.    80.    90.   100.]
 [  900.    90.   100.   110.]
 [ 1000.   100.   110.   120.]]
```

Retrieving domain *y* variable for arbitrary range *x* variable:

```
>>> x = 150
>>> range_ = tstack([np.sin(np.linspace(0, 1, 10))] * 3)
>>> range_ += np.array([0.0, 0.25, 0.5])
>>> MultiSignals(range_, domain)[x]
array([ 0.0359701...,  0.2845447...,  0.5331193...])
>>> x = np.linspace(100, 1000, 3)
```

(continues on next page)

(continued from previous page)

```
>>> MultiSignals(range_, domain)[x]
array([[  4.4085384...e-20,   2.5000000...e-01,   5.0000000...e-01],
       [  4.7669395...e-01,   7.2526859...e-01,   9.7384323...e-01],
       [  8.4147098...e-01,   1.0914709...e+00,   1.3414709...e+00]])
```

Using an alternative interpolating function:

```
>>> x = 150
>>> from colour.algebra import CubicSplineInterpolator
>>> MultiSignals(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
array([ 0.0555274...,  0.3055274...,  0.5555274...])
>>> x = np.linspace(100, 1000, 3)
>>> MultiSignals(
...     range_,
...     domain,
...     interpolator=CubicSplineInterpolator)[x]
array([[ 0.       ...,  0.25     ...,  0.5      ...],
       [ 0.4794253...,  0.7294253...,  0.9794253...],
       [ 0.8414709...,  1.0914709...,  1.3414709...]])
```

arithmetical_operation(*a*, *operation*, *in_place=False*)

Performs given arithmetical operation with a operand, the operation can be either performed on a copy or in-place.

Parameters

- **a** (numeric or ndarray or Signal) – Operand.
- **operation** (object) – Operation to perform.
- **in_place** (bool, optional) – Operation happens in place.

Returns multi-continuous signals.

Return type *MultiSignals*

Examples

Adding a single *numeric* variable:

```
>>> domain = np.arange(0, 10, 1)
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> multi_signals_1 = MultiSignals(range_)
>>> print(multi_signals_1)
[[   0.   10.   20.   30.]
 [   1.   20.   30.   40.]
 [   2.   30.   40.   50.]
 [   3.   40.   50.   60.]
 [   4.   50.   60.   70.]
 [   5.   60.   70.   80.]
 [   6.   70.   80.   90.]
 [   7.   80.   90.  100.]
 [   8.   90.  100.  110.]
 [   9.  100.  110.  120.]]
>>> print(multi_signals_1.arithmetical_operation(10, '+', True))
[[   0.   20.   30.   40.]
 [   1.   30.   40.   50.]
 [   2.   40.   50.   60.]
```

(continues on next page)

(continued from previous page)

```
 [   3.   50.   60.   70.]
 [   4.   60.   70.   80.]
 [   5.   70.   80.   90.]
 [   6.   80.   90.  100.]
 [   7.   90.  100.  110.]
 [   8.  100.  110.  120.]
 [   9.  110.  120.  130.]]
```

Adding an *array_like* variable:

```
>>> a = np.linspace(10, 100, 10)
>>> print(multi_signals_1.arithmetical_operation(a, '+', True))
[[   0.   30.   40.   50.]
 [   1.   50.   60.   70.]
 [   2.   70.   80.   90.]
 [   3.   90.  100.  110.]
 [   4.  110.  120.  130.]
 [   5.  130.  140.  150.]
 [   6.  150.  160.  170.]
 [   7.  170.  180.  190.]
 [   8.  190.  200.  210.]
 [   9.  210.  220.  230.]]
```

```
>>> a = np.array([[10, 20, 30]])
>>> print(multi_signals_1.arithmetical_operation(a, '+', True))
[[   0.   40.   60.   80.]
 [   1.   60.   80.  100.]
 [   2.   80.  100.  120.]
 [   3.  100.  120.  140.]
 [   4.  120.  140.  160.]
 [   5.  140.  160.  180.]
 [   6.  160.  180.  200.]
 [   7.  180.  200.  220.]
 [   8.  200.  220.  240.]
 [   9.  220.  240.  260.]]
```

```
>>> a = np.arange(0, 30, 1).reshape([10, 3])
>>> print(multi_signals_1.arithmetical_operation(a, '+', True))
[[   0.   40.   61.   82.]
 [   1.   63.   84.  105.]
 [   2.   86.  107.  128.]
 [   3.  109.  130.  151.]
 [   4.  132.  153.  174.]
 [   5.  155.  176.  197.]
 [   6.  178.  199.  220.]
 [   7.  201.  222.  243.]
 [   8.  224.  245.  266.]
 [   9.  247.  268.  289.]]
```

Adding a `colour.continuous.Signal` sub-class:

```
>>> multi_signals_2 = MultiSignals(range_)
>>> print(multi_signals_1.arithmetical_operation(
...     multi_signals_2, '+', True))
[[   0.   50.   81.  112.]
 [   1.   83.  114.  145.]
 [   2.  116.  147.  178.]
 [   3.  149.  180.  211.]
 [   4.  182.  213.  244.]
 [   5.  215.  246.  277.]
```

(continues on next page)

(continued from previous page)

```
[   6.  248.  279.  310.]
[   7.  281.  312.  343.]
[   8.  314.  345.  376.]
[   9.  347.  378.  409.]]
```

property domain

Getter and setter property for the colour.continuous.Signal sub-class instances independent domain x variable.

Parameters **value** (array_like) – Value to set the colour.continuous.Signal sub-class instances independent domain x variable with.

Returns colour.continuous.Signal sub-class instances independent domain x variable.

Return type ndarray

property dtype

Getter and setter property for the continuous signal dtype.

Parameters **value** (type) – Value to set the continuous signal dtype with.

Returns Continuous signal dtype.

Return type type

property extrapolator

Getter and setter property for the colour.continuous.Signal sub-class instances extrapolator type.

Parameters **value** (type) – Value to set the colour.continuous.Signal sub-class instances extrapolator type with.

Returns colour.continuous.Signal sub-class instances extrapolator type.

Return type type

property extrapolator_args

Getter and setter property for the colour.continuous.Signal sub-class instances extrapolator instantiation time arguments.

Parameters **value** (dict) – Value to set the colour.continuous.Signal sub-class instances extrapolator instantiation time arguments to.

Returns colour.continuous.Signal sub-class instances extrapolator instantiation time arguments.

Return type dict

fill_nan(*method='Interpolation'*, *default=0*)

Fill NaNs in independent domain x variable and corresponding range y variable using given method.

Parameters

- **method** (unicode, optional) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with default.
- **default** (numeric, optional) – Value to use with the *Constant* method.

Returns

- *Signal* – NaNs filled multi-continuous signals.
- *>>> domain = np.arange(0, 10, 1)*
- *>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)*

- *>>> range_ += np.array([0, 10, 20])*
- *>>> multi_signals = MultiSignals(range_)*
- **>>> multi_signals[3** *(7] = np.nan)*
- *>>> print(multi_signals)*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. nan nan nan] [4. nan nan nan] [5. nan nan nan] [6. nan nan nan] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]
- *>>> print(multi_signals.fill_nan())*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. 40. 50. 60.] [4. 50. 60. 70.] [5. 60. 70. 80.] [6. 70. 80. 90.] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]
- **>>> multi_signals[3** *(7] = np.nan)*
- *>>> print(multi_signals.fill_nan(method='Constant'))*
- *[[0. 10. 20. 30.]* – [1. 20. 30. 40.] [2. 30. 40. 50.] [3. 0. 0. 0.] [4. 0. 0. 0.] [5. 0. 0. 0.] [6. 0. 0. 0.] [7. 80. 90. 100.] [8. 90. 100. 110.] [9. 100. 110. 120.]]

property function

Getter and setter property for the `colour.continuous.Signal` sub-class instances callable.

Parameters **value** (`object`) – Attribute value.

Returns `colour.continuous.Signal` sub-class instances callable.

Return type callable

Notes

- This property is read only.

property interpolator

Getter and setter property for the `colour.continuous.Signal` sub-class instances interpolator type.

Parameters **value** (`type`) – Value to set the `colour.continuous.Signal` sub-class instances interpolator type with.

Returns `colour.continuous.Signal` sub-class instances interpolator type.

Return type type

property interpolator_args

Getter and setter property for the `colour.continuous.Signal` sub-class instances interpolator instantiation time arguments.

Parameters **value** (`dict`) – Value to set the `colour.continuous.Signal` sub-class instances interpolator instantiation time arguments to.

Returns `colour.continuous.Signal` sub-class instances interpolator instantiation time arguments.

Return type dict

property labels

Getter and setter property for the `colour.continuous.Signal` sub-class instances name.

Parameters **value** (`array_like`) – Value to set the `colour.continuous.Signal` sub-class instances name.

Returns `colour.continuous.Signal` sub-class instance name.

Return type dict

static multi_signals_unpack_data(*data=None*, *domain=None*, *labels=None*, *dtype=<class 'numpy.float64'>*, *signal_type=<class 'colour.continuous.signal.Signal'>*, ***kwargs*)

Unpack given data for multi-continuous signals instantiation.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or array_like or dict_like, optional) – Data to unpack for multi-continuous signals instantiation.
- **domain** (array_like, optional) – Values to initialise the multiple colour.continuous.Signal sub-class instances colour.continuous.Signal.domain attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.domain attribute.
- **dtype** (type, optional) – **{np.float16, np.float32, np.float64, np.float128}**, Floating point data type.
- **signal_type** (type, optional) – A colour.continuous.Signal sub-class type.

Other Parameters

- **name** (*unicode, optional*) – multi-continuous signals name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.continuous.Signal sub-class instances.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the colour.continuous.Signal sub-class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the colour.continuous.Signal sub-class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the colour.continuous.Signal sub-class instances.

Returns Mapping of labeled colour.continuous.Signal sub-class instances.

Return type dict

Examples

Unpacking using implicit *domain* and a single signal:

```
>>> range_ = np.linspace(10, 100, 10)
>>> signals = MultiSignals.multi_signals_unpack_data(range_)
>>> list(signals.keys())
[0]
>>> print(signals[0])
[[  0.   10.]
 [  1.   20.]
 [  2.   30.]
 [  3.   40.]
 [  4.   50.]
 [  5.   60.]
 [  6.   70.]
 [  7.   80.]
 [  8.   90.]
 [  9.  100.]]
```

Unpacking using explicit *domain* and a single signal:

```
>>> domain = np.arange(100, 1100, 100)
>>> signals = MultiSignals.multi_signals_unpack_data(range_, domain)
>>> list(signals.keys())
[0]
>>> print(signals[0])
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Unpacking using multiple signals:

```
>>> range_ = tstack([np.linspace(10, 100, 10)] * 3)
>>> range_ += np.array([0, 10, 20])
>>> signals = MultiSignals.multi_signals_unpack_data(range_, domain)
>>> list(signals.keys())
[0, 1, 2]
>>> print(signals[2])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using a *dict*:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_)))
>>> list(signals.keys())
[0, 1, 2]
>>> print(signals[2])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using *MultiSignals.multi_signals_unpack_data* method output:

```
>>> signals = MultiSignals.multi_signals_unpack_data(
...     dict(zip(domain, range_)))
>>> signals = MultiSignals.multi_signals_unpack_data(signals)
>>> list(signals.keys())
[0, 1, 2]
>>> print(signals[2])
[[  100.    30.]
```

(continues on next page)

(continued from previous page)

```
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

Unpacking using a *Pandas Series*:

```
>>> if is_pandas_installed():
...     from pandas import Series
...     signals = MultiSignals.multi_signals_unpack_data(
...         Series(dict(zip(domain, np.linspace(10, 100, 10)))))
...     print(signals[0])
[[  100.    10.]
 [  200.    20.]
 [  300.    30.]
 [  400.    40.]
 [  500.    50.]
 [  600.    60.]
 [  700.    70.]
 [  800.    80.]
 [  900.    90.]
 [ 1000.   100.]]
```

Unpacking using a *Pandas Dataframe*:

```
>>> if is_pandas_installed():
...     from pandas import DataFrame
...     data = dict(zip(['a', 'b', 'c'], tsplit(range_)))
...     signals = MultiSignals.multi_signals_unpack_data(
...         DataFrame(data, domain))
...     print(signals['c'])
[[  100.    30.]
 [  200.    40.]
 [  300.    50.]
 [  400.    60.]
 [  500.    70.]
 [  600.    80.]
 [  700.    90.]
 [  800.   100.]
 [  900.   110.]
 [ 1000.   120.]]
```

property range

Getter and setter property for the `colour.continuous.Signal` sub-class instances corresponding range y variable.

Parameters **value** (`array_like`) – Value to set the `colour.continuous.Signal` sub-class instances corresponding range y variable with.

Returns `colour.continuous.Signal` sub-class instances corresponding range y variable.

Return type ndarray

property signal_type

Getter and setter property for the `colour.continuous.Signal` sub-class instances type.

Returns `colour.continuous.Signal` sub-class instances type.

Return type type

Notes

- This property is read only.

property signals

Getter and setter property for the colour.continuous.Signal sub-class instances.

Parameters **value** (Series or Dataframe or Signal or MultiSignals or array_like or dict_like) – Attribute value.

Returns colour.continuous.Signal sub-class instances.

Return type OrderedDict

to_dataframe()

Converts the continuous signal to a *Pandas* DataFrame class instance.

Returns Continuous signal as a *Pandas* DataFrame class instance.

Return type DataFrame

Examples

```
>>> if is_pandas_installed():
...     domain = np.arange(0, 10, 1)
...     range_ = tstack([np.linspace(10, 100, 10)] * 3)
...     range_ += np.array([0, 10, 20])
...     multi_signals = MultiSignals(range_)
...     print(multi_signals.to_dataframe())
      0      1      2
0.0   10.0   20.0   30.0
1.0   20.0   30.0   40.0
2.0   30.0   40.0   50.0
3.0   40.0   50.0   60.0
4.0   50.0   60.0   70.0
5.0   60.0   70.0   80.0
6.0   70.0   80.0   90.0
7.0   80.0   90.0  100.0
8.0   90.0  100.0  110.0
9.0  100.0  110.0  120.0
```

Corresponding Chromaticities

- *Prediction*
  - *Fairchild (1990)*
  - *CIE 1994*
  - *CMCCAT2000*
  - *Von Kries*

Prediction

colour

corresponding_chromaticities_prediction([...])	Returns the corresponding chromaticities prediction for given chromatic adaptation model.
CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS	Aggregated corresponding chromaticities prediction models.
CorrespondingColourDataset	Defines a corresponding colour dataset.
CorrespondingChromaticitiesPrediction	Defines a chromatic adaptation model prediction.

colour.corresponding_chromaticities_prediction

colour.**corresponding_chromaticities_prediction**(*experiment=1*, *model='Von Kries'*, ***kwargs*)

Returns the corresponding chromaticities prediction for given chromatic adaptation model.

Parameters

- **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **model** (unicode, optional) – **{'Von Kries', 'CIE 1994', 'CMCCAT2000', 'Fairchild 1990'}**, Chromatic adaptation model.

Other Parameters **transform** (*unicode, optional*) – {colour.corresponding.corresponding_chromaticities_prediction_VonKries()}, **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction(2, 'CMCCAT2000')
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[((0.207, 0.486), (0.2083210..., 0.4727168...)),
 ((0.449, 0.511), (0.4459270..., 0.5077735...)),
 ((0.263, 0.505), (0.2640262..., 0.4955361...)),
 ((0.322, 0.545), (0.3316884..., 0.5431580...)),
 ((0.316, 0.537), (0.3222624..., 0.5357624...)),
 ((0.265, 0.553), (0.2710705..., 0.5501997...)),
 ((0.221, 0.538), (0.2261826..., 0.5294740...)),
 ((0.135, 0.532), (0.1439693..., 0.5190984...)),
 ((0.145, 0.472), (0.1494835..., 0.4556760...)),
 ((0.163, 0.331), (0.1563172..., 0.3164151...)),
 ((0.176, 0.431), (0.1763199..., 0.4127589...)),
 ((0.244, 0.349), (0.2287638..., 0.3499324...))]
```

colour.CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS

colour.**CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS = CaseInsensitiveMapping({'CIE 1994': ..., 'CMCCAT**
Aggregated corresponding chromaticities prediction models.

References

[Bre87], [CIET13294], [Fai91], [Fai13c], [Fai13b], [LLRH02], [WRC12a]

CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS [CaseInsensitiveMapping] **{'CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries'}**

Aliases:

- 'vonkries': 'Von Kries'

colour.CorrespondingColourDataset

class colour.**CorrespondingColourDataset**
Defines a corresponding colour dataset.

Parameters

- **name** (unicode) – Corresponding colour dataset name.
- **XYZ_r** (array_like) – *CIE XYZ* tristimulus values of the reference illuminant.
- **XYZ_t** (array_like) – *CIE XYZ* tristimulus values of the test illuminant.
- **XYZ_cr** (array_like) – Corresponding *CIE XYZ* tristimulus values under the reference illuminant.
- **XYZ_ct** (array_like) – Corresponding *CIE XYZ* tristimulus values under the test illuminant.
- **Y_r** (numeric) – Reference white luminance Y_r in cd/m^2.
- **Y_t** (numeric) – Test white luminance Y_t in cd/m^2.
- **B_r** (numeric) – Luminance factor B_r of reference achromatic background as percentage.
- **B_t** (numeric) – Luminance factor B_t of test achromatic background as percentage.
- **metadata** (dict) – Dataset metadata.

Notes

- This class is compatible with *Luo and Rhodes (1999) Corresponding-Colour Datasets* datasets.

References

[LR99]

Create new instance of CorrespondingColourDataset(name, XYZ_r, XYZ_t, XYZ_cr, XYZ_ct, Y_r, Y_t, B_r, B_t, metadata)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

B_r	Alias for field number 7
B_t	Alias for field number 8
XYZ_cr	Alias for field number 3
XYZ_ct	Alias for field number 4
XYZ_r	Alias for field number 1
XYZ_t	Alias for field number 2
Y_r	Alias for field number 5
Y_t	Alias for field number 6
metadata	Alias for field number 9
name	Alias for field number 0

colour.CorrespondingChromaticitiesPrediction

class colour.**CorrespondingChromaticitiesPrediction**

Defines a chromatic adaptation model prediction.

Parameters

- **name** (unicode) – Test colour name.
- **uv_t** (array_like, (2,)) – Chromaticity coordinates uv_t^p of test colour.
- **uv_m** (array_like, (2,)) – Chromaticity coordinates uv_m^p of matching colour.
- **uv_p** (array_like, (2,)) – Chromaticity coordinates uv_p^p of predicted colour.

Create new instance of CorrespondingChromaticitiesPrediction(name, uv_t, uv_m, uv_p)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

name	Alias for field number 0
uv_m	Alias for field number 2
uv_p	Alias for field number 3
uv_t	Alias for field number 1

Dataset

colour

BRENEMAN_EXPERIMENTS	*Breneman (1987)* experiments.
BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES	*Breneman (1987)* experiments primaries chromaticities.

colour.BRENEMAN_EXPERIMENTS

colour.**BRENEMAN_EXPERIMENTS = {1: (BrenemanExperimentResult(name='Illuminant', uv_t=array([0.259, 0.526]**

corresponding_chromaticities_prediction_Fairchild1990(...)	Returns the corresponding chromaticities prediction for *Fairchild (1990)* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_Fairchild1990

colour.corresponding.**corresponding_chromaticities_prediction_Fairchild1990**(*experiment=1*)

Returns the corresponding chromaticities prediction for *Fairchild (1990)* chromatic adaptation model.

Parameters **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [Fai91], [Fai13c]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_Fairchild1990(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2089528...,  0.4724034...])),
 (array([ 0.449,  0.511]), array([ 0.4375652...,  0.5121030...])),
 (array([ 0.263,  0.505]), array([ 0.2621362...,  0.4972538...])),
 (array([ 0.322,  0.545]), array([ 0.3235312...,  0.5475665...])),
 (array([ 0.316,  0.537]), array([ 0.3151391...,  0.5398333...])),
 (array([ 0.265,  0.553]), array([ 0.2634745...,  0.5544335...])),
 (array([ 0.221,  0.538]), array([ 0.2211595...,  0.5324470...])),
 (array([ 0.135,  0.532]), array([ 0.1396949...,  0.5207234...])),
 (array([ 0.145,  0.472]), array([ 0.1512288...,  0.4533041...])),
 (array([ 0.163,  0.331]), array([ 0.1715691...,  0.3026264...])),
 (array([ 0.176,  0.431]), array([ 0.1825792...,  0.4077892...])),
 (array([ 0.244,  0.349]), array([ 0.2418905...,  0.3413401...]))]
```

CIE 1994

colour.corresponding

corresponding_chromaticities_prediction_CIE1994([...])	Returns the corresponding chromaticities prediction for *CIE 1994* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_CIE1994

colour.corresponding.**corresponding_chromaticities_prediction_CIE1994**(*experiment=1*)

Returns the corresponding chromaticities prediction for *CIE 1994* chromatic adaptation model.

Parameters

- **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance. Returns
- **-------** –
- **tuple** – Corresponding chromaticities prediction.

References

[Bre87], [CIET13294]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CIE1994(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2273130...,  0.5267609...])),
 (array([ 0.449,  0.511]), array([ 0.4612181...,  0.5191849...])),
 (array([ 0.263,  0.505]), array([ 0.2872404...,  0.5306938...])),
 (array([ 0.322,  0.545]), array([ 0.3489822...,  0.5454398...])),
 (array([ 0.316,  0.537]), array([ 0.3371612...,  0.5421567...])),
 (array([ 0.265,  0.553]), array([ 0.2889416...,  0.5534074...])),
 (array([ 0.221,  0.538]), array([ 0.2412195...,  0.5464301...])),
 (array([ 0.135,  0.532]), array([ 0.1530344...,  0.5488239...])),
 (array([ 0.145,  0.472]), array([ 0.1568709...,  0.5258835...])),
 (array([ 0.163,  0.331]), array([ 0.1499762...,  0.4401747...])),
 (array([ 0.176,  0.431]), array([ 0.1876711...,  0.5039627...])),
 (array([ 0.244,  0.349]), array([ 0.2560012...,  0.4546263...]))]
```

CMCCAT2000

colour.corresponding

corresponding_chromaticities_prediction_CMCCAT2000([...])	Returns the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

colour.corresponding.corresponding_chromaticities_prediction_CMCCAT2000

colour.corresponding.**corresponding_chromaticities_prediction_CMCCAT2000**(*experiment=1*)

Returns the corresponding chromaticities prediction for *CMCCAT2000* chromatic adaptation model.

Parameters **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [LLRH02], [WRC12a]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_CMCCAT2000(2)
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2083210...,  0.4727168...])),
 (array([ 0.449,  0.511]), array([ 0.4459270...,  0.5077735...])),
 (array([ 0.263,  0.505]), array([ 0.2640262...,  0.4955361...])),
 (array([ 0.322,  0.545]), array([ 0.3316884...,  0.5431580...])),
 (array([ 0.316,  0.537]), array([ 0.3222624...,  0.5357624...])),
 (array([ 0.265,  0.553]), array([ 0.2710705...,  0.5501997...])),
 (array([ 0.221,  0.538]), array([ 0.2261826...,  0.5294740...])),
 (array([ 0.135,  0.532]), array([ 0.1439693...,  0.5190984...])),
 (array([ 0.145,  0.472]), array([ 0.1494835...,  0.4556760...])),
 (array([ 0.163,  0.331]), array([ 0.1563172...,  0.3164151...])),
 (array([ 0.176,  0.431]), array([ 0.1763199...,  0.4127589...])),
 (array([ 0.244,  0.349]), array([ 0.2287638...,  0.3499324...]))]
```

Von Kries

colour.corresponding

corresponding_chromaticities_prediction_VonKries([...])	Returns the corresponding chromaticities prediction for *Von Kries* chromatic adaptation model using given transform.

colour.corresponding.corresponding_chromaticities_prediction_VonKries

colour.corresponding.**corresponding_chromaticities_prediction_VonKries**(*experiment=1*, *transform='CAT02'*)

Returns the corresponding chromaticities prediction for *Von Kries* chromatic adaptation model using given transform.

Parameters

- **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, Chromatic adaptation transform.

Returns Corresponding chromaticities prediction.

Return type tuple

References

[Bre87], [Fai13b]

Examples

```
>>> from pprint import pprint
>>> pr = corresponding_chromaticities_prediction_VonKries(2, 'Bradford')
>>> pr = [(p.uv_m, p.uv_p) for p in pr]
>>> pprint(pr)
[(array([ 0.207,  0.486]), array([ 0.2082014...,  0.4722922...])),
 (array([ 0.449,  0.511]), array([ 0.4489102...,  0.5071602...])),
 (array([ 0.263,  0.505]), array([ 0.2643545...,  0.4959631...])),
 (array([ 0.322,  0.545]), array([ 0.3348730...,  0.5471220...])),
 (array([ 0.316,  0.537]), array([ 0.3248758...,  0.5390589...])),
 (array([ 0.265,  0.553]), array([ 0.2733105...,  0.5555028...])),
 (array([ 0.221,  0.538]), array([ 0.227148 ...,  0.5331318...])),
 (array([ 0.135,  0.532]), array([ 0.1442730...,  0.5226804...])),
 (array([ 0.145,  0.472]), array([ 0.1498745...,  0.4550785...])),
 (array([ 0.163,  0.331]), array([ 0.1564975...,  0.3148796...])),
 (array([ 0.176,  0.431]), array([ 0.1760593...,  0.4103772...])),
 (array([ 0.244,  0.349]), array([ 0.2259805...,  0.3465291...]))]
```

Colour Difference

- *Delta E*
- *CIE 1976*
- *CIE 1994*
- *CIE 2000*
- *CMC*
- *Luo, Cui and Li (2006)*
- *Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)*
- *DIN99*

Delta E

colour

delta_E(a, b[, method])	Returns the difference ΔE_{ab} between two given *CIE L*a*b** or $J'a'b'$ colourspace arrays using given method.
DELTA_E_METHODS	Supported ΔE_{ab} computation methods.

colour.delta_E

colour.**delta_E**(*a*, *b*, *method='CIE 2000'*, ***kwargs*)

Returns the difference ΔE_{ab} between two given *CIE L*a*b** or $J'a'b'$ colourspace arrays using given method.

Parameters

- **a** (array_like) – *CIE L*a*b** or $J'a'b'$ colourspace array a.
- **b** (array_like) – *CIE L*a*b** or $J'a'b'$ colourspace array b.
- **method** (unicode, optional) – **{'CIE 2000', 'CIE 1976', 'CIE 1994', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS', 'DIN99'}** Computation method.

Other Parameters

- **textiles** (*bool, optional*) – {colour.difference.delta_E_CIE1994(), colour.difference.delta_E_CIE2000(), colour.difference.delta_E_DIN99()}, Textiles application specific parametric factors $k_L = 2$, $k_C = k_H = 1$, $k_1 = 0.048$, $k_2 = 0.014$, $k_E = 2$, $k_CH = 0.5$ weights are used instead of $k_L = k_C = k_H = 1$, $k_1 = 0.045$, $k_2 = 0.015$, $k_E = k_CH = 1.0$.
- **l** (*numeric, optional*) – {colour.difference.delta_E_CIE2000()}, Lightness weighting factor.
- **c** (*numeric, optional*) – {colour.difference.delta_E_CIE2000()}, Chroma weighting factor.

Returns Colour difference ΔE_{ab}.

Return type numeric or ndarray

References

[ASTMInternational07], [LLW+17], [Lin03a], [Lin11], [Lin09b], [Lin09c], [LCL06], [Mel13], [Wik08c]

Examples

```
>>> import numpy as np
>>> a = np.array([100.00000000, 21.57210357, 272.22819350])
>>> b = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E(a, b)
94.0356490...
>>> delta_E(a, b, method='CIE 2000')
94.0356490...
>>> delta_E(a, b, method='CIE 1976')
451.7133019...
>>> delta_E(a, b, method='CIE 1994')
83.7792255...
>>> delta_E(a, b, method='CIE 1994', textiles=False)
...
83.7792255...
>>> delta_E(a, b, method='DIN99')
66.1119282...
>>> a = np.array([54.90433134, -0.08450395, -0.06854831])
>>> b = np.array([54.90433134, -0.08442362, -0.06848314])
>>> delta_E(a, b, method='CAM02-UCS')
0.0001034...
```

(continues on next page)

(continued from previous page)

```
>>> delta_E(a, b, method='CAM16-LCD')
0.0001034...
```

colour.DELTA_E_METHODS

colour.**DELTA_E_METHODS = CaseInsensitiveMapping({'CIE 1976': ..., 'CIE 1994': ..., 'CIE 2000': ..., 'CM**

Supported ΔE_{ab} computation methods.

References

[ASTMInternational07], [LLW+17], [Lin03a], [Lin11], [Lin09b], [Lin09c], [LCL06], [Mel13], [Wik08c]

DELTA_E_METHODS [CaseInsensitiveMapping] **{'CIE 1976', 'CIE 1994', 'CIE 2000', 'CMC', 'CAM02-LCD', 'CAM02-SCD', 'CAM02-UCS', 'CAM16-LCD', 'CAM16-SCD', 'CAM16-UCS', 'DIN99'}**

Aliases:

- 'cie1976': 'CIE 1976'
- 'cie1994': 'CIE 1994'
- 'cie2000': 'CIE 2000'

CIE 1976

colour.difference

delta_E_CIE1976(Lab_1, Lab_2)	Returns the difference ΔE_{76} between two given *CIE L*a*b** colourspace arrays using *CIE 1976* recommendation.

colour.difference.delta_E_CIE1976

colour.difference.**delta_E_CIE1976**(*Lab_1*, *Lab_2*)

Returns the difference ΔE_{76} between two given *CIE L*a*b** colourspace arrays using *CIE 1976* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace array 2.

Returns Colour difference ΔE_{76}.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[Lin03a]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE1976(Lab_1, Lab_2)
451.7133019...
```

CIE 1994

colour.difference

delta_E_CIE1994(Lab_1, Lab_2[, textiles])	Returns the difference ΔE_{94} between two given *CIE L*a*b** colourspace arrays using *CIE 1994* recommendation.

colour.difference.delta_E_CIE1994

colour.difference.**delta_E_CIE1994**(*Lab_1*, *Lab_2*, *textiles=False*)

Returns the difference ΔE_{94} between two given *CIE L*a*b** colourspace arrays using *CIE 1994* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace array 2.
- **textiles** (bool, optional) – Textiles application specific parametric factors $k_L = 2,\ k_C = k_H = 1,\ k_1 = 0.048,\ k_2 = 0.014$ weights are used instead of $k_L = k_C = k_H = 1,\ k_1 = 0.045,\ k_2 = 0.015$.

Returns Colour difference ΔE_{94}.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

- *CIE 1994* colour differences are not symmetrical: difference between Lab_1 and Lab_2 may not be the same as difference between Lab_2 and Lab_1 thus one colour must be understood to be the reference against which a sample colour is compared.

References

[Lin11]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE1994(Lab_1, Lab_2)
83.7792255...
>>> delta_E_CIE1994(Lab_1, Lab_2, textiles=True)
88.3355530...
```

CIE 2000

colour.difference

delta_E_CIE2000(Lab_1, Lab_2[, textiles])	Returns the difference ΔE_{00} between two given *CIE L*a*b** colourspace arrays using *CIE 2000* recommendation.

colour.difference.delta_E_CIE2000

colour.difference.**delta_E_CIE2000**(*Lab_1*, *Lab_2*, *textiles=False*)

Returns the difference ΔE_{00} between two given *CIE L*a*b** colourspace arrays using *CIE 2000* recommendation.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace array 2.
- **textiles** (bool, optional) – Textiles application specific parametric factors $k_L = 2,\ k_C = k_H = 1$ weights are used instead of $k_L = k_C = k_H = 1$.

Returns Colour difference ΔE_{00}.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

- *CIE 2000* colour differences are not symmetrical: difference between Lab_1 and Lab_2 may not be the same as difference between Lab_2 and Lab_1 thus one colour must be understood to be the reference against which a sample colour is compared.
- Parametric factors $k_L = k_C = k_H = 1$ weights under *reference conditions*:
  - Illumination: D65 source
  - Illuminance: 1000 lx
  - Observer: Normal colour vision
  - Background field: Uniform, neutral gray with $L^* = 50$
  - Viewing mode: Object
  - Sample size: Greater than 4 degrees
  - Sample separation: Direct edge contact
  - Sample colour-difference magnitude: Lower than 5.0 ΔE_{00}
  - Sample structure: Homogeneous (without texture)

References

[Lin09b], [Mel13]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE2000(Lab_1, Lab_2)
94.0356490...
>>> Lab_2 = np.array([50.00000000, 426.67945353, 72.39590835])
>>> delta_E_CIE2000(Lab_1, Lab_2)
100.8779470...
>>> delta_E_CIE2000(Lab_1, Lab_2, textiles=True)
95.7920535...
```

CMC

colour.difference

delta_E_CMC(Lab_1, Lab_2[, l, c])	Returns the difference ΔE_{CMC} between two given *CIE L*a*b** colourspace arrays using *Colour Measurement Committee* recommendation.

colour.difference.delta_E_CMC

colour.difference.**delta_E_CMC**(*Lab_1*, *Lab_2*, *l=2*, *c=1*)

Returns the difference ΔE_{CMC} between two given *CIE L*a*b** colourspace arrays using *Colour Measurement Committee* recommendation.

The quasimetric has two parameters: *Lightness* (l) and *chroma* (c), allowing the users to weight the difference based on the ratio of l:c. Commonly used values are 2:1 for acceptability and 1:1 for the threshold of imperceptibility.

Parameters

- **Lab_1** (array_like) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (array_like) – *CIE L*a*b** colourspace array 2.
- **l** (numeric, optional) – Lightness weighting factor.
- **c** (numeric, optional) – Chroma weighting factor.

Returns Colour difference ΔE_{CMC}.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_1	L_1 : [0, 100] a_1 : [-100, 100] b_1 : [-100, 100]	L_1 : [0, 1] a_1 : [-1, 1] b_1 : [-1, 1]
Lab_2	L_2 : [0, 100] a_2 : [-100, 100] b_2 : [-100, 100]	L_2 : [0, 1] a_2 : [-1, 1] b_2 : [-1, 1]

References

[Lin09c]

Examples

```
>>> Lab_1 = np.array([100.00000000, 21.57210357, 272.22819350])
>>> Lab_2 = np.array([100.00000000, 426.67945353, 72.39590835])
>>> delta_E_CMC(Lab_1, Lab_2)
172.7047712...
```

Luo, Cui and Li (2006)

colour.difference

delta_E_CAM02LCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et al.(2017) CAM16-LCD* colourspaces $J'a'b'$ arrays..
delta_E_CAM02SCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et al.(2017) CAM16-SCD* colourspaces $J'a'b'$ arrays..
delta_E_CAM02UCS(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et al.(2017) CAM16-UCS* colourspaces $J'a'b'$ arrays..

colour.difference.delta_E_CAM02LCD

colour.difference.**delta_E_CAM02LCD**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02LCD()` definition for an usage example.

colour.difference.delta_E_CAM02SCD

`colour.difference.`**`delta_E_CAM02SCD`**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (`array_like`) – Standard / reference *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (`array_like`) – Sample / test *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02SCD()` definition for an usage example.

colour.difference.delta_E_CAM02UCS

colour.difference.**delta_E_CAM02UCS**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to colour.delta_E_CAM02UCS() definition for an usage example.

Li, Li, Wang, Zu, Luo, Cui, Melgosa, Brill and Pointer (2017)

colour.difference

delta_E_CAM16LCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et al.(2017) CAM16-LCD* colourspaces $J'a'b'$ arrays..
delta_E_CAM16SCD(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et al.(2017) CAM16-SCD* colourspaces $J'a'b'$ arrays..
delta_E_CAM16UCS(Jpapbp_1, Jpapbp_2)	Returns the difference $\Delta E'$ between two given *Li et al.(2017) CAM16-UCS* colourspaces $J'a'b'$ arrays..

colour.difference.delta_E_CAM16LCD

colour.difference.**delta_E_CAM16LCD**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-LCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to colour.delta_E_CAM02LCD() definition for an usage example.

colour.difference.delta_E_CAM16SCD

colour.difference.**delta_E_CAM16SCD**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-SCD* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02SCD()` definition for an usage example.

colour.difference.delta_E_CAM16UCS

colour.difference.**delta_E_CAM16UCS**(*Jpapbp_1*, *Jpapbp_2*)

Returns the difference $\Delta E'$ between two given *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ arrays.

Parameters

- **Jpapbp_1** (array_like) – Standard / reference *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.
- **Jpapbp_2** (array_like) – Sample / test *Li et al. (2017) CAM16-UCS* colourspaces $J'a'b'$ array.

Returns Colour difference $\Delta E'$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp_1	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]
Jpapbp_2	Jp_2 : [0, 100] ap_2 : [-100, 100] bp_2 : [-100, 100]	Jp_2 : [0, 1] ap_2 : [-1, 1] bp_2 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.delta_E_CAM02UCS()` definition for an usage example.

DIN99

`colour.difference`

`delta_E_DIN99`(Lab_1, Lab_2[, textiles])	Returns the difference ΔE_{DIN99} between two given *CIE L*a*b** colourspace arrays using *DIN99* formula.

colour.difference.delta_E_DIN99

`colour.difference.`**`delta_E_DIN99`**(*Lab_1*, *Lab_2*, *textiles=False*)

Returns the difference ΔE_{DIN99} between two given *CIE L*a*b** colourspace arrays using *DIN99* formula.

Parameters

- **Lab_1** (`array_like`) – *CIE L*a*b** colourspace array 1.
- **Lab_2** (`array_like`) – *CIE L*a*b** colourspace array 2.

Returns Colour difference ΔE_{DIN99}.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
`Lab_1`	`L_1` : [0, 100] `a_1` : [-100, 100] `b_1` : [-100, 100]	`L_1` : [0, 1] `a_1` : [-1, 1] `b_1` : [-1, 1]
`Lab_2`	`L_2` : [0, 100] `a_2` : [-100, 100] `b_2` : [-100, 100]	`L_2` : [0, 1] `a_2` : [-1, 1] `b_2` : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab_1 = np.array([60.2574, -34.0099, 36.2677])
>>> Lab_2 = np.array([60.4626, -34.1751, 39.4387])
>>> delta_E_DIN99(Lab_1, Lab_2)
1.1772166...
```

Automatic Colour Conversion Graph

- *Conversion*

Conversion

colour

convert(a, source, target, **kwargs)	Converts given object a from source colour representation to target colour representation using the automatic colour conversion graph.
describe_conversion_path(source, target[, ...])	Describes the conversion path from source colour representation to target colour representation using the automatic colour conversion graph.

colour.convert

colour.**convert**(*a*, *source*, *target*, ***kwargs*)

Converts given object a from source colour representation to target colour representation using the automatic colour conversion graph.

The conversion is performed by finding the shortest path in a NetworkX DiGraph class instance.

The conversion path adopts the **'1'** domain-range scale and the object a is expected to be *soft* normalised accordingly. For example, *CIE XYZ* tristimulus values arguments for use with the *CAM16* colour appearance model should be in domain *[0, 1]* instead of the domain *[0, 100]* used with the **'Reference'** domain-range scale. The arguments are typically converted as follows:

- *Scalars* in domain-range *[0, 10]*, e.g *Munsell Value* are scaled by *10*.
- *Percentages* in domain-range *[0, 100]* are scaled by *100*.
- *Degrees* in domain-range *[0, 360]* are scaled by *360*.
- *Integers* in domain-range *[0, 2**n -1]* where *n* is the bit depth are scaled by *2**n -1*.

See the Domain-Range Scales page for more information.

Parameters

- **a** (array_like or numeric or SpectralDistribution) – Object a to convert. If a represents a reflectance, transmittance or absorptance value, the expectation is that it is viewed under *CIE Standard Illuminant D Series D65*. The illuminant can be changed on a per definition basis along the conversion path.
- **source** (unicode) – Source colour representation, i.e. the source node in the automatic colour conversion graph.

- **target** (unicode) – Target colour representation, i.e. the target node in the automatic colour conversion graph.

Other Parameters ****kwargs** (*dict, optional*) – {'*'}, Please refer to the documentation of the supported conversion definitions.

Arguments for the conversion definitions are passed as keyword arguments whose names is those of the conversion definitions and values set as dictionaries. For example, in the conversion from spectral distribution to *sRGB* colourspace, passing arguments to the `colour.sd_to_XYZ()` definition is done as follows:

```
convert(sd, 'Spectral Distribution', 'sRGB', sd_to_XYZ={'illuminant':␣
↪ILLUMINANTS_SDS['FL2']})
```

It is also possible to pass keyword arguments directly to the various conversion definitions irrespective of their name. This is dangerous and could cause unexpected behaviour because of unavoidable discrepancies with the underlying `colour.utilities.filter_kwargs()` definition between Python 2.7 and 3.x. Using this direct keyword arguments passing mechanism might also ends up passing incompatible arguments to a given conversion definition. Consider the following conversion:

```
convert(sd, 'Spectral Distribution', 'sRGB', 'illuminant': ILLUMINANTS_SDS['FL2
↪'])
```

Because both the `colour.sd_to_XYZ()` and `colour.XYZ_to_sRGB()` definitions have an *illuminant* argument, *ILLUMINANTS_SDS['FL2']* will be passed to both of them and will raise an exception in the `colour.XYZ_to_sRGB()` definition. This will be addressed in the future by either catching the exception and trying a new time without the keyword argument or more elegantly via type checking.

With that in mind, this mechanism offers some good benefits: For example, it allows defining a conversion from *CIE XYZ* colourspace to *n* different colour models while passing an illuminant argument but without having to explicitly define all the explicit conversion definition arguments:

```
a = np.array([0.20654008, 0.12197225, 0.05136952])
illuminant = ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']
for model in ('CIE xyY', 'CIE Lab'):
    convert(a, 'CIE XYZ', model, illuminant=illuminant)
```

Instead of:

```
for model in ('CIE xyY', 'CIE Lab'):
    convert(a, 'CIE XYZ', model, XYZ_to_xyY={'illuminant': illuminant}, XYZ_to_
↪Lab={'illuminant': illuminant})
```

Mixing both approaches is possible for the brevity benefits. It is made possible because the keyword arguments directly passed are filtered first and then the resulting dict is updated with the explicit conversion definition arguments:

```
illuminant = ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']
 convert(sd, 'Spectral Distribution', 'sRGB', 'illuminant': ILLUMINANTS_SDS['FL2
↪'], XYZ_to_sRGB={'illuminant': illuminant})
```

For inspection purposes, verbose is enabled by passing arguments to the `colour.describe_conversion_path()` definition via the verbose keyword argument as follows:

```
convert(sd, 'Spectral Distribution', 'sRGB', verbose={'mode': 'Long'})
```

Returns Converted object a.

Return type ndarray or numeric or *SpectralDistribution*

Warning: The domain-range scale is **'1'** and cannot be changed.

Notes

- Various defaults have been systematically adopted compared to the low-level *Colour* API:
  - The default illuminant for the computation is *CIE Standard Illuminant D Series D65*. It can be changed on a per definition basis along the conversion path.
  - The default *RGB* colourspace is the *sRGB* colourspace. It can also be changed on a per definition basis along the conversion path.
  - Most of the colour appearance models have defaults set according to *IEC 61966-2-1:1999* viewing conditions, i.e. *sRGB* 64 Lux ambiant illumination, 80 cd/m^2, adapting field luminance about 20% of a white object in the scene.
- The **RGB** colour representation is assumed to be linear and representing *scene-referred* imagery. To convert such *RGB* values to *output-referred* (*display-referred*) imagery, i.e. encode the *RGB* values using an encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF), the **Output-Referred RGB** representation must be used.

Examples

```
>>> from colour import COLOURCHECKERS_SDS
>>> sd = COLOURCHECKERS_SDS['ColorChecker N Ohta']['dark skin']
>>> convert(sd, 'Spectral Distribution', 'sRGB',
...     verbose={'mode': 'Short', 'width': 75})
...
================================================================================
*                                                                              *
*   [ Conversion Path ]                                                        *
*                                                                              *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                              *
*                                                                              *
================================================================================
array([ 0.4567579...,  0.3098698...,  0.2486192...])
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> convert(sd, 'Spectral Distribution', 'sRGB',
...     sd_to_XYZ={'illuminant': illuminant})
...
array([ 0.4792457...,  0.3167696...,  0.1736272...])
>>> a = np.array([0.45675795, 0.30986982, 0.24861924])
>>> convert(a, 'Output-Referred RGB', 'CAM16UCS')
...
array([ 0.3999481...,  0.0920655...,  0.0812752...])
>>> a = np.array([0.39994811, 0.09206558, 0.08127526])
>>> convert(a, 'CAM16UCS', 'sRGB', verbose={'mode': 'Short', 'width': 75})
...
================================================================================
*                                                                              *
*   [ Conversion Path ]                                                        *
*                                                                              *
*   "UCS_Luo2006_to_JMh_CIECAM02" --> "JMh_CAM16_to_CAM16" -->                 *
*   "CAM16_to_XYZ" --> "XYZ_to_sRGB"                                           *
*                                                                              *
================================================================================
array([ 0.4567576...,  0.3098826...,  0.2486222...])
```

colour.describe_conversion_path

colour.**describe_conversion_path**(*source*, *target*, *mode='Short'*, *width=79*, *padding=3*, *print_callable=<built-in function print>*, ***kwargs*)

Describes the conversion path from source colour representation to target colour representation using the automatic colour conversion graph.

Parameters

- **source** (unicode) – Source colour representation, i.e. the source node in the automatic colour conversion graph.
- **target** (unicode) – Target colour representation, i.e. the target node in the automatic colour conversion graph.
- **mode** (unicode, optional) – **{'Short', 'Long', 'Extended'}**, Verbose mode: *Short* describes the conversion path, *Long* provides details about the arguments, definitions signatures and output values, *Extended* appends the definitions documentation.
- **width** (int, optional) – Message box width.
- **padding** (unicode, optional) – Padding on each sides of the message.
- **print_callable** (callable, optional) – Callable used to print the message box.

Other Parameters ****kwargs** (*dict, optional*) – {colour.convert()}, Please refer to the documentation of the previously listed definition.

Examples

```
>>> describe_conversion_path('Spectral Distribution', 'sRGB', width=75)
===========================================================================
*                                                                         *
*   [ Conversion Path ]                                                   *
*                                                                         *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                         *
*                                                                         *
===========================================================================
```

Input and Output

- *Image Data*
- *Look Up Table (LUT) Data*
- *CSV Tabular Data*
- *IES TM-27-14 Data*
- *X-Rite Data*

Image Data

colour

READ_IMAGE_METHODS	Supported read image methods.
read_image(path[, bit_depth, method])	Reads the image at given path using given method.
WRITE_IMAGE_METHODS	Supported write image methods.
write_image(image, path[, bit_depth, method])	Writes given image at given path using given method.

colour.READ_IMAGE_METHODS

colour.**READ_IMAGE_METHODS = CaseInsensitiveMapping({'Imageio': ..., 'OpenImageIO': ...})**

Supported read image methods.

READ_IMAGE_METHODS [CaseInsensitiveMapping] **{'Imageio', 'OpenImageIO'}**

colour.read_image

colour.**read_image**(*path*, *bit_depth='float32'*, *method='OpenImageIO'*, ***kwargs*)

Reads the image at given path using given method.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Returned image bit depth, for the *Imageio* method, the image data is converted with colour.io.convert_bit_depth() definition after reading the image, for the *OpenImageIO* method, the bit depth conversion behaviour is driven directly by the library, this definition only converts to the relevant data type after reading.
- **method** (unicode, optional) – **{'OpenImageIO', 'Imageio'}**, Read method, i.e. the image library used for reading images.

Other Parameters **attributes** (*bool, optional*) – {colour.io.read_image_OpenImageIO()}, Whether to return the image attributes.

Returns Image as a ndarray.

Return type ndarray

Notes

- If the given method is *OpenImageIO* but the library is not available writing will be performed by *Imageio*.
- If the given method is *Imageio*, kwargs is passed directly to the wrapped definition.
- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> image.shape
(1267, 1274, 3)
>>> image.dtype
dtype('float32')
```

colour.WRITE_IMAGE_METHODS

colour.**WRITE_IMAGE_METHODS = CaseInsensitiveMapping({'Imageio': ..., 'OpenImageIO': ...})**
Supported write image methods.

WRITE_IMAGE_METHODS [CaseInsensitiveMapping] **{'Imageio', 'OpenImageIO'}**

colour.write_image

colour.**write_image**(*image*, *path*, *bit_depth='float32'*, *method='OpenImageIO'*, ***kwargs*)
Writes given image at given path using given method.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Bit depth to write the image at, for the *Imageio* method, the image data is converted with colour.io.convert_bit_depth() definition prior to writing the image.
- **method** (unicode, optional) – **{'OpenImageIO', 'Imageio'}**, Write method, i.e. the image library used for writing images.

Other Parameters **attributes** (*array_like, optional*) – {colour.io.write_image_OpenImageIO()}, An array of colour.io.ImageAttribute_Specification class instances used to set attributes of the image.

Returns Definition success.

Return type bool

Notes

- If the given method is *OpenImageIO* but the library is not available writing will be performed by *Imageio*.
- If the given method is *Imageio*, kwargs is passed directly to the wrapped definition.

Examples

Basic image writing:

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMSTestPattern.tif')
>>> write_image(image, path)
True
```

Advanced image writing while setting attributes using *OpenImageIO*:

```
>>> compression = ImageAttribute_Specification('Compression', 'none')
>>> write_image(image, path, bit_depth='uint8', attributes=[compression])
...
True
```

Ancillary Objects

`colour.io`

ImageAttribute_Specification	Defines the an image specification attribute.
convert_bit_depth(a[, bit_depth])	Converts given array to given bit depth, the current bit depth of the array is used to determine the appropriate conversion path.
read_image_OpenImageIO(path[, bit_depth, ...])	Reads the image at given path using *OpenImageIO*.
write_image_OpenImageIO(image, path[, ...])	Writes given image at given path using *OpenImageIO*.
read_image_Imageio(path[, bit_depth])	Reads the image at given path using *Imageio*.
write_image_Imageio(image, path[, bit_depth])	Writes given image at given path using *Imageio*.

colour.io.ImageAttribute_Specification

class colour.io.**ImageAttribute_Specification**

Defines the an image specification attribute.

Parameters

- **name** (unicode) – Attribute name.
- **value** (object) – Attribute value.
- **type_** (TypeDesc, optional) – Attribute type as an *OpenImageIO* TypeDesc class instance.

Returns a new instance of the `colour.ImageAttribute_Specification` class.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`	Initialize self.
`count(value)`	
`index(value, [start, [stop]])`	Raises ValueError if the value is not present.

Attributes

`name`	Alias for field number 0
`type_`	Alias for field number 2
`value`	Alias for field number 1

colour.io.convert_bit_depth

`colour.io.`**`convert_bit_depth`**(*a*, *bit_depth='float32'*)

Converts given array to given bit depth, the current bit depth of the array is used to determine the appropriate conversion path.

Parameters

- **a** (`array_like`) – Array to convert to given bit depth.
- **bit_depth** (`unicode`) – Bit depth.

Returns Converted array.

Return type ndarray

Examples

```
>>> a = np.array([0.0, 0.5, 1.0])
>>> convert_bit_depth(a, 'uint8')
array([  0, 128, 255], dtype=uint8)
>>> convert_bit_depth(a, 'uint16')
array([    0, 32768, 65535], dtype=uint16)
>>> convert_bit_depth(a, 'float16')
array([ 0. ,  0.5,  1. ], dtype=float16)
>>> a = np.array([0, 128, 255], dtype=np.uint8)
>>> convert_bit_depth(a, 'uint16')
array([    0, 32896, 65535], dtype=uint16)
>>> convert_bit_depth(a, 'float32')
array([ 0.       ,  0.501960...,  1.       ], dtype=float32)
```

colour.io.read_image_OpenImageIO

`colour.io.`**`read_image_OpenImageIO`**(*path*, *bit_depth='float32'*, *attributes=False*)

Reads the image at given path using *OpenImageIO*.

Parameters

- **path** (`unicode`) – Image path.
- **bit_depth** (`unicode, optional`) – **{'float32', 'uint8', 'uint16', 'float16'}**, Returned image bit depth, the bit depth conversion behaviour is driven directly by *OpenImageIO*, this definition only converts to the relevant data type after reading.

- **attributes** (bool, optional) – Whether to return the image attributes.

Returns Image as a ndarray or tuple of image as ndarray and list of attributes

Return type ndarray or tuple

Notes

- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
```

colour.io.write_image_OpenImageIO

colour.io.**write_image_OpenImageIO**(*image*, *path*, *bit_depth='float32'*, *attributes=None*)

Writes given image at given path using *OpenImageIO*.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Bit depth to write the image at, the bit depth conversion behaviour is ruled directly by *OpenImageIO*.
- **attributes** (array_like, optional) – An array of colour.io.ImageAttribute_Specification class instances used to set attributes of the image.

Returns Definition success.

Return type bool

Examples

Basic image writing:

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMSTestPattern.tif')
>>> write_image(image, path)
True
```

Advanced image writing while setting attributes:

```
>>> compression = ImageAttribute_Specification('Compression', 'none')
>>> write_image(image, path, 'uint8', [compression])
True
```

colour.io.read_image_Imageio

colour.io.**read_image_Imageio**(*path*, *bit_depth='float32'*, ***kwargs*)

Reads the image at given path using *Imageio*.

Parameters

- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Returned image bit depth, the image data is converted with colour.io.convert_bit_depth() definition after reading the image.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Image as a ndarray.

Return type ndarray

Notes

- For convenience, single channel images are squeezed to 2d arrays.

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image_Imageio(path)
>>> image.shape
(1267, 1274, 3)
>>> image.dtype
dtype('float32')
```

colour.io.write_image_Imageio

colour.io.**write_image_Imageio**(*image*, *path*, *bit_depth='float32'*, ***kwargs*)

Writes given image at given path using *Imageio*.

Parameters

- **image** (array_like) – Image data.
- **path** (unicode) – Image path.
- **bit_depth** (unicode, optional) – **{'float32', 'uint8', 'uint16', 'float16'}**, Bit depth to write the image at, the image data is converted with colour.io.convert_bit_depth() definition prior to writing the image.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Definition success.

Return type bool

Examples

```
>>> import os
>>> import colour
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMS_Test_Pattern.exr')
>>> image = read_image(path)
>>> path = os.path.join(colour.__path__[0], 'io', 'tests', 'resources',
...                     'CMSTestPattern.tif')
>>> write_image(image, path)
True
```

Look Up Table (LUT) Data

colour

LUT1D([table, name, domain, size, comments])	Defines the base class for a 1D *LUT*.
LUT3x1D([table, name, domain, size, comments])	Defines the base class for a 3x1D *LUT*.
LUT3D([table, name, domain, size, comments])	Defines the base class for a 3D *LUT*.
LUTSequence(*args)	Defines the base class for a *LUT* sequence, i.e.

colour.LUT1D

class colour.**LUT1D**(*table=None*, *name=None*, *domain=None*, *size=10*, *comments=None*)

Bases: colour.io.luts.lut.AbstractLUT

Defines the base class for a 1D *LUT*.

Parameters

- **table** (array_like, optional) – Underlying *LUT* table.
- **name** (unicode, optional) – *LUT* name.
- **domain** (unicode, optional) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (int, optional) – Size of the instantiation time default table.
- **comments** (array_like, optional) – Comments to add to the *LUT*.

is_domain_explicit()

linear_table()

apply()

as_LUT()

Examples

Instantiating a unity LUT with a table with 16 elements:

```
>>> print(LUT1D(size=16))
LUT1D - Unity 16
----------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (16,)
```

Instantiating a LUT using a custom table with 16 elements:

```
>>> print(LUT1D(LUT1D.linear_table(16) ** (1 / 2.2)))
LUT1D - ...
--------...

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (16,)
```

Instantiating a LUT using a custom table with 16 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([-0.1, 1.5])
>>> print(LUT1D(
...     spow(LUT1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.']))
LUT1D - My LUT
--------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

apply(*RGB*, *interpolator=<class 'colour.algebra.interpolation.LinearInterpolator'>*, *interpolator_args=None*)

Applies the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (`array_like`) – *RGB* colourspace array to apply the *LUT* onto.
- **interpolator** (`object, optional`) – Interpolator class type to use as interpolating function.
- **interpolator_args** (`dict_like, optional`) – Arguments to use when instantiating the interpolating function.

Returns Interpolated *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT = LUT1D(LUT1D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4529220...,  0.4529220...,  0.4529220...])
```

as_LUT(*cls*, *force_conversion=False*, ***kwargs*)

Converts the *LUT* to given `cls` class instance.

Parameters

- **cls** (`LUT1D or LUT3x1D or LUT3D`) – *LUT* class instance.
- **force_conversion** (`bool, optional`) – Whether to force the conversion as it might be destructive.

Other Parameters

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **size** (*int, optional*) – Expected table size in case of an upcast to a `LUT3D` class instance.

Returns Converted *LUT* class instance.

Return type *LUT1D* or *LUT3x1D* or *LUT3D*

Warning: Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Examples

```
>>> LUT = LUT1D()
>>> print(LUT.as_LUT(LUT1D))
LUT1D - Unity 10 - Converted 1D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.as_LUT(LUT3x1D))
LUT3x1D - Unity 10 - Converted 1D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.as_LUT(LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 1D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

is_domain_explicit()

Returns whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[0 1]
```

While an explicit domain defines every single discrete samples:

```
[0.0 0.1 0.2 0.4 0.8 1.0]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT1D().is_domain_explicit()
False
>>> table = domain = np.linspace(0, 1, 10)
>>> LUT1D(table, domain=domain).is_domain_explicit()
True
```

static linear_table(*size=10*, *domain=array([0, 1])*)

Returns a linear table, the number of output samples n is equal to `size`.

Parameters

- **size** (`int, optional`) – Expected table size.
- **domain** (`array_like, optional`) – Domain of the table.

Returns Linear table with `size` samples.

Return type ndarray

Examples

```
>>> LUT1D.linear_table(5, np.array([-0.1, 1.5]))
array([-0.1,  0.3,  0.7,  1.1,  1.5])
>>> LUT1D.linear_table(domain=np.linspace(-0.1, 1.5, 5))
array([-0.1,  0.3,  0.7,  1.1,  1.5])
```

colour.LUT3x1D

class colour.**LUT3x1D**(*table=None*, *name=None*, *domain=None*, *size=10*, *comments=None*)

Bases: `colour.io.luts.lut.AbstractLUT`

Defines the base class for a 3x1D *LUT*.

Parameters

- **table** (`array_like, optional`) – Underlying *LUT* table.
- **name** (`unicode, optional`) – *LUT* name.
- **domain** (`unicode, optional`) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (`int, optional`) – Size of the instantiation time default table.
- **comments** (`array_like, optional`) – Comments to add to the *LUT*.

is_domain_explicit()

linear_table()

apply()

as_LUT()

Examples

Instantiating a unity LUT with a table with 16x3 elements:

```
>>> print(LUT3x1D(size=16))
LUT3x1D - Unity 16
------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 3)
```

Instantiating a LUT using a custom table with 16x3 elements:

```
>>> print(LUT3x1D(LUT3x1D.linear_table(16) ** (1 / 2.2)))
...
LUT3x1D - ...
----------...

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 3)
```

Instantiating a LUT using a custom table with 16x3 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> print(LUT3x1D(
...     spow(LUT3x1D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.']))
LUT3x1D - My LUT
----------------

Dimensions : 2
Domain     : [[-0.1 -0.2 -0.4]
              [ 1.5  3.   6. ]]
Size       : (16, 3)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

apply(*RGB*, *interpolator=<class 'colour.algebra.interpolation.LinearInterpolator'>*, *interpolator_args=None*)

Applies the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (`array_like`) – *RGB* colourspace array to apply the *LUT* onto.
- **interpolator** (`object, optional`) – Interpolator class type to use as interpolating function.
- **interpolator_args** (`dict_like, optional`) – Arguments to use when instantiating the interpolating function.

Returns Interpolated *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT = LUT3x1D(LUT3x1D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4529220...,  0.4529220...,  0.4529220...])
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> table = spow(LUT3x1D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3x1D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4423903...,  0.4503801...,  0.3581625...])
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.3, 1.4, 6.0],
...                    [0.7, 3.0, np.nan],
...                    [1.1, np.nan, np.nan],
...                    [1.5, np.nan, np.nan]])
>>> table = spow(LUT3x1D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3x1D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.2996370..., -0.0901332..., -0.3949770...])
```

as_LUT(*cls*, *force_conversion=False*, ***kwargs*)

Converts the *LUT* to given `cls` class instance.

Parameters

- **cls** (`LUT1D or LUT3x1D or LUT3D`) – *LUT* class instance.
- **force_conversion** (`bool, optional`) – Whether to force the conversion as it might be destructive.

Other Parameters

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **size** (*int, optional*) – Expected table size in case of an upcast to a `LUT3D` class instance.

Returns Converted *LUT* class instance.

Return type *LUT1D* or *LUT3x1D* or *LUT3D*

Warning: Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Examples

```
>>> LUT = LUT3x1D()
>>> print(LUT.as_LUT(LUT1D, force_conversion=True))
LUT1D - Unity 10 - Converted 3x1D to 1D
---------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.as_LUT(LUT3x1D))
LUT3x1D - Unity 10 - Converted 3x1D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.as_LUT(LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 3x1D to 3D
---------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

is_domain_explicit()

Returns whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[[0 1]
 [0 1]
 [0 1]]
```

While an explicit domain defines every single discrete samples:

```
[[0.0 0.0 0.0]
 [0.1 0.1 0.1]
 [0.2 0.2 0.2]
 [0.3 0.3 0.3]
 [0.4 0.4 0.4]
 [0.8 0.8 0.8]
 [1.0 1.0 1.0]]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT3x1D().is_domain_explicit()
False
>>> samples = np.linspace(0, 1, 10)
>>> table = domain = tstack([samples, samples, samples])
>>> LUT3x1D(table, domain=domain).is_domain_explicit()
True
```

static linear_table(*size=10, domain=array([[0, 0, 0], [1, 1, 1]])*)

Returns a linear table, the number of output samples n is equal to `size * 3` or `size[0] +`

size[1] + size[2].

Parameters

- **size** (int or array_like, optional) – Expected table size.
- **domain** (array_like, optional) – Domain of the table.

Returns Linear table with size * 3 or size[0] + size[1] + size[2] samples.

Return type ndarray

Warning: If size is non uniform, the linear table will be padded accordingly.

Examples

```
>>> LUT3x1D.linear_table(
...     5, np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[-0.1, -0.2, -0.4],
       [ 0.3,  0.6,  1.2],
       [ 0.7,  1.4,  2.8],
       [ 1.1,  2.2,  4.4],
       [ 1.5,  3. ,  6. ]])
>>> LUT3x1D.linear_table(
...     np.array([5, 3, 2]),
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[-0.1, -0.2, -0.4],
       [ 0.3,  1.4,  6. ],
       [ 0.7,  3. ,  nan],
       [ 1.1,  nan,  nan],
       [ 1.5,  nan,  nan]])
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.3, 1.4, 6.0],
...                    [0.7, 3.0, np.nan],
...                    [1.1, np.nan, np.nan],
...                    [1.5, np.nan, np.nan]])
>>> LUT3x1D.linear_table(domain=domain)
array([[-0.1, -0.2, -0.4],
       [ 0.3,  1.4,  6. ],
       [ 0.7,  3. ,  nan],
       [ 1.1,  nan,  nan],
       [ 1.5,  nan,  nan]])
```

colour.LUT3D

class colour.**LUT3D**(*table=None*, *name=None*, *domain=None*, *size=33*, *comments=None*)

Bases: colour.io.luts.lut.AbstractLUT

Defines the base class for a 3D *LUT*.

Parameters

- **table** (array_like, optional) – Underlying *LUT* table.
- **name** (unicode, optional) – *LUT* name.
- **domain** (unicode, optional) – *LUT* domain, also used to define the instantiation time default table domain.
- **size** (int, optional) – Size of the instantiation time default table.
- **comments** (array_like, optional) – Comments to add to the *LUT*.

is_domain_explicit()

linear_table()

apply()

as_LUT()

Examples

Instantiating a unity LUT with a table with 16x16x16x3 elements:

```
>>> print(LUT3D(size=16))
LUT3D - Unity 16
----------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 16, 16, 3)
```

Instantiating a LUT using a custom table with 16x16x16x3 elements:

```
>>> print(LUT3D(LUT3D.linear_table(16) ** (1 / 2.2)))
LUT3D - ...
--------...

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (16, 16, 16, 3)
```

Instantiating a LUT using a custom table with 16x16x16x3 elements, custom name, custom domain and comments:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> print(LUT3D(
...     spow(LUT3D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.']))
LUT3D - My LUT
--------------

Dimensions : 3
Domain     : [[-0.1 -0.2 -0.4]
              [ 1.5  3.   6. ]]
Size       : (16, 16, 16, 3)
Comment 01 : A first comment.
Comment 02 : A second comment.
```

apply(*RGB*, *interpolator=<function table_interpolation_trilinear>*, *interpolator_args=None*)

Applies the *LUT* to given *RGB* colourspace array using given method.

Parameters

- **RGB** (`array_like`) – *RGB* colourspace array to apply the *LUT* onto.
- **interpolator** (`object, optional`) – Interpolator object to use as interpolating function.
- **interpolator_args** (`dict_like, optional`) – Arguments to use when calling the interpolating function.

Returns Interpolated *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT = LUT3D(LUT3D.linear_table() ** (1 / 2.2))
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.4583277...,  0.4583277...,  0.4583277...])
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.3, 1.4, 6.0],
...                    [0.7, 3.0, np.nan],
...                    [1.1, np.nan, np.nan],
...                    [1.5, np.nan, np.nan]])
>>> table = spow(LUT3D.linear_table(domain=domain), 1 / 2.2)
>>> LUT = LUT3D(table, domain=domain)
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> LUT.apply(RGB)
array([ 0.2996370..., -0.0901332..., -0.3949770...])
```

as_LUT(*cls*, *force_conversion=False*, ***kwargs*)

Converts the *LUT* to given `cls` class instance.

Parameters

- **cls** (`LUT1D or LUT3x1D or LUT3D`) – *LUT* class instance.
- **force_conversion** (`bool, optional`) – Whether to force the conversion as it might be destructive.

Other Parameters

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **size** (*int, optional*) – Expected table size in case of a downcast from a `LUT3D` class instance.

Returns Converted *LUT* class instance.

Return type *LUT1D* or *LUT3x1D* or *LUT3D*

Warning: Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Examples

```
>>> LUT = LUT3D()
>>> print(LUT.as_LUT(LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT.as_LUT(LUT3x1D, force_conversion=True))
LUT3x1D - Unity 33 - Converted 3D to 3x1D
-----------------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
>>> print(LUT.as_LUT(LUT3D))
LUT3D - Unity 33 - Converted 3D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
```

is_domain_explicit()

Returns whether the *LUT* domain is explicit (or implicit).

An implicit domain is defined by its shape only:

```
[[0 0 0]
 [1 1 1]]
```

While an explicit domain defines every single discrete samples:

```
[[0.0 0.0 0.0]
 [0.1 0.1 0.1]
 [0.2 0.2 0.2]
 [0.3 0.3 0.3]
 [0.4 0.4 0.4]
 [0.8 0.8 0.8]
 [1.0 1.0 1.0]]
```

Returns Is *LUT* domain explicit.

Return type bool

Examples

```
>>> LUT3D().is_domain_explicit()
False
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.7, 1.4, 6.0],
...                    [1.5, 3.0, np.nan]])
>>> LUT3D(domain=domain).is_domain_explicit()
True
```

static linear_table(*size=33, domain=array([[0, 0, 0], [1, 1, 1]])*)

Returns a linear table, the number of output samples n is equal to `size**3 * 3` or `size[0] *`

`size[1] * size[2] * 3`.

Parameters

- **size** (`int or array_like, optional`) – Expected table size.
- **domain** (`array_like, optional`) – Domain of the table.

Returns Linear table with `size**3 * 3` or `size[0] * size[1] * size[2] * 3` samples.

Return type ndarray

Examples

```
>>> LUT3D.linear_table(
...     3, np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  2.8],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  2.8],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  2.8],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  2.8],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  2.8],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  2.8],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  2.8],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  2.8],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  2.8],
         [ 1.5,  3. ,  6. ]]]])
>>> LUT3D.linear_table(
...     np.array([3, 3, 2]),
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]))
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
```

(continues on next page)

(continued from previous page)

```
        [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  6. ]]]])
>>> domain = np.array([[-0.1, -0.2, -0.4],
...                    [0.7, 1.4, 6.0],
...                    [1.5, 3.0, np.nan]])
>>> LUT3D.linear_table(domain=domain)
array([[[[-0.1, -0.2, -0.4],
         [-0.1, -0.2,  6. ]],

        [[-0.1,  1.4, -0.4],
         [-0.1,  1.4,  6. ]],

        [[-0.1,  3. , -0.4],
         [-0.1,  3. ,  6. ]]],


       [[[ 0.7, -0.2, -0.4],
         [ 0.7, -0.2,  6. ]],

        [[ 0.7,  1.4, -0.4],
         [ 0.7,  1.4,  6. ]],

        [[ 0.7,  3. , -0.4],
         [ 0.7,  3. ,  6. ]]],


       [[[ 1.5, -0.2, -0.4],
         [ 1.5, -0.2,  6. ]],

        [[ 1.5,  1.4, -0.4],
         [ 1.5,  1.4,  6. ]],

        [[ 1.5,  3. , -0.4],
         [ 1.5,  3. ,  6. ]]]])
```

colour.LUTSequence

class colour.**LUTSequence**(**args*)

Bases: collections.abc.MutableSequence

Defines the base class for a *LUT* sequence, i.e. a series of *LUTs*.

The *colour.LUTSequence* class can be used to model series of *LUTs* such as when a shaper *LUT* is combined with a 3D *LUT*.

Other Parameters ***args** (*list, optional*) – Sequence of *colour.LUT1D*, *colour.LUT3x1D*, *colour.LUT3D* or *colour.io.lut.l.AbstractLUTSequenceOperator* class instances.

sequence

__getitem__()

__setitem__()

__delitem__()

__len__()

__str__()

__repr__()

__eq__()

__ne__()

insert()

apply()

copy()

Examples

```
>>> LUT_1 = LUT1D()
>>> LUT_2 = LUT3D(size=3)
>>> LUT_3 = LUT3x1D()
>>> print(LUTSequence(LUT_1, LUT_2, LUT_3))
LUT Sequence
------------

Overview

    LUT1D ---> LUT3D ---> LUT3x1D

Operations

    LUT1D - Unity 10
    ----------------

    Dimensions : 1
    Domain     : [ 0.  1.]
    Size       : (10,)

    LUT3D - Unity 3
    ---------------

    Dimensions : 3
    Domain     : [[ 0.  0.  0.]
                  [ 1.  1.  1.]]
    Size       : (3, 3, 3, 3)
```

(continues on next page)

(continued from previous page)

```
LUT3x1D - Unity 10
------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (10, 3)
```

apply(*RGB*, *interpolator_1D=<class 'colour.algebra.interpolation.LinearInterpolator'>*, *interpolator_1D_args=None*, *interpolator_3D=<function table_interpolation_trilinear>*, *interpolator_3D_args=None*)

Applies the *LUT* sequence sequentially to given *RGB* colourspace array.

Parameters

- **RGB** (array_like) – *RGB* colourspace array to apply the *LUT* sequence sequentially onto.
- **interpolator_1D** (object, optional) – Interpolator object to use as interpolating function for colour.LUT1D (and colour.LUT3x1D) class instances.
- **interpolator_1D_args** (dict_like, optional) – Arguments to use when calling the interpolating function for colour.LUT1D (and colour.LUT3x1D) class instances.
- **interpolator_3D** (object, optional) – Interpolator object to use as interpolating function for colour.LUT3D class instances.
- **interpolator_3D_args** (dict_like, optional) – Arguments to use when calling the interpolating function for colour.LUT3D class instances.

Returns Processed *RGB* colourspace array.

Return type ndarray

Examples

```
>>> LUT_1 = LUT1D(LUT1D.linear_table(16) + 0.125)
>>> LUT_2 = LUT3D(LUT3D.linear_table(16) ** (1 / 2.2))
>>> LUT_3 = LUT3x1D(LUT3x1D.linear_table(16) * 0.750)
>>> LUT_sequence = LUTSequence(LUT_1, LUT_2, LUT_3)
>>> samples = np.linspace(0, 1, 5)
>>> RGB = tstack([samples, samples, samples])
>>> LUT_sequence.apply(RGB)
array([[ 0.2899886...,  0.2899886...,  0.2899886...],
       [ 0.4797662...,  0.4797662...,  0.4797662...],
       [ 0.6055328...,  0.6055328...,  0.6055328...],
       [ 0.7057779...,  0.7057779...,  0.7057779...],
       [ 0.75     ...,  0.75     ...,  0.75     ...]])
```

copy()

Returns a copy of the *LUT* sequence.

Returns *LUT* sequence copy.

Return type *LUTSequence*

insert(*index*, *LUT*)

Inserts given *LUT* at given index into the *LUT* sequence.

Parameters

- **index** (index) – Index to insert the *LUT* at into the *LUT* sequence.

- **LUT** (LUT1D or LUT3x1D or LUT3D or AbstractLUTSequenceOperator) – *LUT* to insert into the *LUT* sequence.

property sequence

Getter and setter property for the underlying *LUT* sequence.

Parameters **value** (list) – Value to set the the underlying *LUT* sequence with.

Returns Underlying *LUT* sequence.

Return type list

read_LUT(path[, method])	Reads given *LUT* file using given method.
write_LUT(LUT, path[, decimals, method])	Writes given *LUT* to given file using given method.

colour.read_LUT

colour.**read_LUT**(*path*, *method=None*, ***kwargs*)

Reads given *LUT* file using given method.

Parameters

- **path** (unicode) – *LUT* path.
- **method** (unicode, optional) – **{None, 'Cinespace', 'Iridas Cube', 'Resolve Cube', 'Sony SPI1D', 'Sony SPI3D'}**, Reading method, if *None*, the method will be auto-detected according to extension.

Returns LUT1D, LUT3x1D or LUT3D class instance.

Return type *LUT1D* or *LUT3x1D* or *LUT3D*

References

[AdobeSystems13c], [Cha15], [RisingSResearch]

Examples

Reading a 3x1D *Iridas .cube LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'ACES_Proxy_10_to_ACES.cube')
>>> print(read_LUT(path))
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'eotf_sRGB_1D.spi1d')
>>> print(read_LUT(path))
LUT1D - eotf sRGB 1D
--------------------
```

(continues on next page)

(continued from previous page)

```
Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

Reading a 3D *Sony .spi3d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'Colour_Correct.spi3d')
>>> print(read_LUT(path))
LUT3D - Colour Correct
----------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
```

colour.write_LUT

colour.**write_LUT**(*LUT*, *path*, *decimals=7*, *method=None*, ***kwargs*)

Writes given *LUT* to given file using given method.

Parameters

- **LUT** (LUT1D or LUT3x1D or LUT3D) – LUT1D, LUT3x1D or LUT3D class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.
- **method** (unicode, optional) – **{None, 'Cinespace', 'Iridas Cube', 'Resolve Cube', 'Sony SPI1D', 'Sony SPI3D'}**, Writing method, if *None*, the method will be auto-detected according to extension.

Returns Definition success.

Return type bool

References

[AdobeSystems13c], [Cha15], [RisingSResearch]

Examples

Writing a 3x1D *Iridas .cube LUT*:

```
>>> import numpy as np
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
```

(continues on next page)

(continued from previous page)

```
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT(LUT, 'My_LUT.cube')
```

Writing a 1D *Sony .spi1d LUT*:

```
>>> domain = np.array([-0.1, 1.5])
>>> LUT = LUT1D(
...     spow(LUT1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT(LUT, 'My_LUT.spi1d')
```

Writing a 3D *Sony .spi3d LUT*:

```
>>> LUT = LUT3D(
...     LUT3D.linear_table(16) ** (1 / 2.2),
...     'My LUT',
...     np.array([[0, 0, 0], [1, 1, 1]]),
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT(LUT, 'My_LUT.cube')
```

Ancillary Objects

`colour.io`

AbstractLUTSequenceOperator	Defines the base class for *LUT* sequence operators.

colour.io.AbstractLUTSequenceOperator

class colour.io.**AbstractLUTSequenceOperator**

Bases: object

Defines the base class for *LUT* sequence operators.

This is an ABCMeta abstract class that must be inherited by sub-classes.

apply()

abstract apply(*RGB*, **args*)

Applies the *LUT* sequence operator to given *RGB* colourspace array.

Parameters **RGB** (array_like) – *RGB* colourspace array to apply the *LUT* sequence operator onto.

Returns Processed *RGB* colourspace array.

Return type ndarray

LUT_to_LUT(LUT, cls[, force_conversion])	Converts given *LUT* to given cls class instance.
read_LUT_Cinespace(path)	Reads given *Cinespace .csp LUT* file.
write_LUT_Cinespace(LUT, path[, decimals])	Writes given *LUT* to given *Cinespace .csp LUT* file.
read_LUT_IridasCube(path)	Reads given *Iridas .cube LUT* file.
write_LUT_IridasCube(LUT, path[, decimals])	Writes given *LUT* to given *Iridas .cube LUT* file.
read_LUT_SonySPI1D(path)	Reads given *Sony .spi1d LUT* file.
write_LUT_SonySPI1D(LUT, path[, decimals])	Writes given *LUT* to given *Sony .spi1d LUT* file.
read_LUT_SonySPI3D(path)	Reads given *Sony .spi3d LUT* file.
write_LUT_SonySPI3D(LUT, path[, decimals])	Writes given *LUT* to given *Sony .spi3d LUT* file.

colour.io.LUT_to_LUT

colour.io.**LUT_to_LUT**(*LUT*, *cls*, *force_conversion=False*, ***kwargs*)

Converts given *LUT* to given `cls` class instance.

Parameters

- **cls** (`LUT1D or LUT3x1D or LUT3D`) – *LUT* class instance.
- **force_conversion** (`bool, optional`) – Whether to force the conversion as it might be destructive.

Other Parameters

- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function.
- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function.
- **size** (*int, optional*) – Expected table size in case of an upcast to or a downcast from a `LUT3D` class instance.
- **channel_weights** (*array_like, optional*) – Channel weights in case of a downcast from a `LUT3x1D` or `LUT3D` class instance.

Returns Converted *LUT* class instance.

Return type *LUT1D* or *LUT3x1D* or *LUT3D*

Warning: Some conversions are destructive and raise a `ValueError` exception by default.

Raises `ValueError` – If the conversion is destructive.

Examples

```
>>> print(LUT_to_LUT(LUT1D(), LUT3D, force_conversion=True))
LUT3D - Unity 10 - Converted 1D to 3D
-------------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (33, 33, 33, 3)
>>> print(LUT_to_LUT(LUT3x1D(), LUT1D, force_conversion=True))
LUT1D - Unity 10 - Converted 3x1D to 1D
---------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
>>> print(LUT_to_LUT(LUT3D(), LUT1D, force_conversion=True))
LUT1D - Unity 33 - Converted 3D to 1D
-------------------------------------

Dimensions : 1
Domain     : [ 0.  1.]
Size       : (10,)
```

colour.io.read_LUT_Cinespace

colour.io.**read_LUT_Cinespace**(*path*)

Reads given *Cinespace .csp LUT* file.

Parameters **path** (unicode) – *LUT* path.

Returns LUT3x1D or LUT3D or LUTSequence class instance.

Return type *LUT3x1D* or *LUT3D* or *LUTSequence*

References

[RisingSResearch]

Examples

Reading a 3x1D *Cinespace .csp LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'cinespace',
...     'ACES_Proxy_10_to_ACES.csp')
>>> print(read_LUT_Cinespace(path))
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 3D *Cinespace .csp LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'cinespace',
...     'Colour_Correct.csp')
>>> print(read_LUT_Cinespace(path))
LUT3D - Generated by Foundry::LUT
---------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
```

colour.io.write_LUT_Cinespace

colour.io.**write_LUT_Cinespace**(*LUT*, *path*, *decimals=7*)

Writes given *LUT* to given *Cinespace .csp LUT* file.

Parameters

- **LUT** (LUT1D or LUT3x1D or LUT3D or LUTSequence) – LUT1D, LUT3x1D or LUT3D or LUTSequence class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type bool

References

[RisingSResearch]

Examples

Writing a 3x1D *Cinespace .csp LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_Cinespace(LUT, 'My_LUT.cube')
```

Writing a 3D *Cinespace .csp LUT*:

```
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3D(
...     spow(LUT3D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_Cinespace(LUT, 'My_LUT.cube')
```

colour.io.read_LUT_IridasCube

colour.io.**read_LUT_IridasCube**(*path*)

Reads given *Iridas .cube LUT* file.

Parameters **path** (unicode) – *LUT* path.

Returns LUT3x1D or LUT3D class instance.

Return type *LUT3x1D* or LUT3d

References

[AdobeSystems13c]

Examples

Reading a 3x1D *Iridas .cube LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'ACES_Proxy_10_to_ACES.cube')
>>> print(read_LUT_IridasCube(path))
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------

Dimensions : 2
```

(continues on next page)

(continued from previous page)

```
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (32, 3)
```

Reading a 3D *Iridas .cube LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'Colour_Correct.cube')
>>> print(read_LUT_IridasCube(path))
LUT3D - Generated by Foundry::LUT
---------------------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
```

Reading a 3D *Iridas .cube LUT* with comments:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'iridas_cube',
...     'Demo.cube')
>>> print(read_LUT_IridasCube(path))
LUT3x1D - Demo
--------------

Dimensions : 2
Domain     : [[ 0.  0.  0.]
              [ 1.  2.  3.]]
Size       : (3, 3)
Comment 01 : Comments can go anywhere
```

colour.io.write_LUT_IridasCube

colour.io.**write_LUT_IridasCube**(*LUT*, *path*, *decimals=7*)

Writes given *LUT* to given *Iridas .cube LUT* file.

Parameters

- **LUT** (LUT3x1D or LUT3d or LUTSequence) – LUT3x1D, LUT3D or LUTSequence class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type bool

Warning:

- If a LUTSequence class instance is passed as LUT, the first *LUT* in the *LUT* sequence will be used.

References

[AdobeSystems13c]

Examples

Writing a 3x1D *Iridas .cube LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_IridasCube(LUT, 'My_LUT.cube')
```

Writing a 3D *Iridas .cube LUT*:

```
>>> domain = np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]])
>>> LUT = LUT3D(
...     spow(LUT3D.linear_table(16, domain), 1 / 2.2),
...     'My LUT',
...     np.array([[-0.1, -0.2, -0.4], [1.5, 3.0, 6.0]]),
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_IridasCube(LUT, 'My_LUT.cube')
```

colour.io.read_LUT_SonySPI1D

colour.io.**read_LUT_SonySPI1D**(*path*)

Reads given *Sony .spi1d LUT* file.

Parameters **path** (unicode) – *LUT* path.

Returns LUT1D or LUT3x1D class instance.

Return type *LUT1D* or *LUT3x1D*

Examples

Reading a 1D *Sony .spi1d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'eotf_sRGB_1D.spi1d')
>>> print(read_LUT_SonySPI1D(path))
LUT1D - eotf sRGB 1D
--------------------

Dimensions : 1
Domain     : [-0.1  1.5]
Size       : (16,)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

Reading a 3x1D *Sony .spi1d LUT*:

```
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi1d',
...     'eotf_sRGB_3x1D.spi1d')
>>> print(read_LUT_SonySPI1D(path))
LUT3x1D - eotf sRGB 3x1D
------------------------

Dimensions : 2
Domain     : [[-0.1 -0.1 -0.1]
              [ 1.5  1.5  1.5]]
Size       : (16, 3)
Comment 01 : Generated by "Colour 0.3.11".
Comment 02 : "colour.models.eotf_sRGB".
```

colour.io.write_LUT_SonySPI1D

colour.io.**write_LUT_SonySPI1D**(*LUT*, *path*, *decimals=7*)

Writes given *LUT* to given *Sony .spi1d LUT* file.

Parameters

- **LUT** (LUT1D or LUT2d) – LUT1D, LUT3x1D or LUTSequence class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type bool

Warning:

- If a LUTSequence class instance is passed as LUT, the first *LUT* in the *LUT* sequence will be used.

Examples

Writing a 1D *Sony .spi1d LUT*:

```
>>> from colour.algebra import spow
>>> domain = np.array([-0.1, 1.5])
>>> LUT = LUT1D(
...     spow(LUT1D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI1D(LUT, 'My_LUT.cube')
```

Writing a 3x1D *Sony .spi1d LUT*:

```
>>> domain = np.array([[-0.1, -0.1, -0.1], [1.5, 1.5, 1.5]])
>>> LUT = LUT3x1D(
...     spow(LUT3x1D.linear_table(16), 1 / 2.2),
...     'My LUT',
...     domain,
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI1D(LUT, 'My_LUT.cube')
```

colour.io.read_LUT_SonySPI3D

colour.io.**read_LUT_SonySPI3D**(*path*)
Reads given *Sony .spi3d LUT* file.

Parameters **path** (unicode) – *LUT* path.

Returns LUT3D or LUT3x1D class instance.

Return type *LUT3D* or *LUT3x1D*

Examples

Reading a 3D *Sony .spi3d LUT*:

```
>>> import os
>>> path = os.path.join(
...     os.path.dirname(__file__), 'tests', 'resources', 'sony_spi3d',
...     'Colour_Correct.spi3d')
>>> print(read_LUT_SonySPI3D(path))
LUT3D - Colour Correct
----------------------

Dimensions : 3
Domain     : [[ 0.  0.  0.]
              [ 1.  1.  1.]]
Size       : (4, 4, 4, 3)
Comment 01 : Adapted from a LUT generated by Foundry::LUT.
```

colour.io.write_LUT_SonySPI3D

colour.io.**write_LUT_SonySPI3D**(*LUT*, *path*, *decimals=7*)
Writes given *LUT* to given *Sony .spi3d LUT* file.

Parameters

- **LUT** (LUT3D) – LUT3D or LUTSequence class instance to write at given path.
- **path** (unicode) – *LUT* path.
- **decimals** (int, optional) – Formatting decimals.

Returns Definition success.

Return type bool

Warning:

- If a LUTSequence class instance is passed as LUT, the first *LUT* in the *LUT* sequence will be used.

Examples

Writing a 3D *Sony .spi3d LUT*:

```
>>> LUT = LUT3D(
...     LUT3D.linear_table(16) ** (1 / 2.2),
...     'My LUT',
...     np.array([[0, 0, 0], [1, 1, 1]]),
...     comments=['A first comment.', 'A second comment.'])
>>> write_LUT_SonySPI3D(LUT, 'My_LUT.cube')
```

CSV Tabular Data

colour

read_sds_from_csv_file(path[, delimiter, ...])	Reads the spectral data from given *CSV* file and return its content as an *OrderedDict* of colour.SpectralDistribution classes.
read_spectral_data_from_csv_file(path[, ...])	Reads the spectral data from given *CSV* file in the following form:
write_sds_to_csv_file(sds, path[, ...])	Writes the given spectral distributions to given *CSV* file.

colour.read_sds_from_csv_file

colour.**read_sds_from_csv_file**(*path*, *delimiter=', '*, *fields=None*, *default=0*)

Reads the spectral data from given *CSV* file and return its content as an *OrderedDict* of colour.SpectralDistribution classes.

Parameters

- **path** (unicode) – Absolute *CSV* file path.
- **delimiter** (unicode, optional) – *CSV* file content delimiter.
- **fields** (array_like, optional) – *CSV* file spectral data fields names. If no value is provided the first line of the file will be used for as spectral data fields names.
- **default** (numeric) – Default value for fields row with missing value.

Returns colour.SpectralDistribution classes of given *CSV* file.

Return type OrderedDict

Examples

```
>>> from colour.utilities import numpy_print_options
>>> import os
>>> csv_file = os.path.join(os.path.dirname(__file__), 'tests',
...                         'resources', 'colorchecker_n_ohta.csv')
>>> sds = read_sds_from_csv_file(csv_file)
>>> print(tuple(sds.keys()))
('1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
↪'18', '19', '20', '21', '22', '23', '24')
>>> with numpy_print_options(suppress=True):
...     sds['1']
SpectralDistribution([[ 380.   ,    0.048],
                      [ 385.   ,    0.051],
```

(continues on next page)

(continued from previous page)

```
                    [ 390.   ,    0.055],
                    [ 395.   ,    0.06 ],
                    [ 400.   ,    0.065],
                    [ 405.   ,    0.068],
                    [ 410.   ,    0.068],
                    [ 415.   ,    0.067],
                    [ 420.   ,    0.064],
                    [ 425.   ,    0.062],
                    [ 430.   ,    0.059],
                    [ 435.   ,    0.057],
                    [ 440.   ,    0.055],
                    [ 445.   ,    0.054],
                    [ 450.   ,    0.053],
                    [ 455.   ,    0.053],
                    [ 460.   ,    0.052],
                    [ 465.   ,    0.052],
                    [ 470.   ,    0.052],
                    [ 475.   ,    0.053],
                    [ 480.   ,    0.054],
                    [ 485.   ,    0.055],
                    [ 490.   ,    0.057],
                    [ 495.   ,    0.059],
                    [ 500.   ,    0.061],
                    [ 505.   ,    0.062],
                    [ 510.   ,    0.065],
                    [ 515.   ,    0.067],
                    [ 520.   ,    0.07 ],
                    [ 525.   ,    0.072],
                    [ 530.   ,    0.074],
                    [ 535.   ,    0.075],
                    [ 540.   ,    0.076],
                    [ 545.   ,    0.078],
                    [ 550.   ,    0.079],
                    [ 555.   ,    0.082],
                    [ 560.   ,    0.087],
                    [ 565.   ,    0.092],
                    [ 570.   ,    0.1  ],
                    [ 575.   ,    0.107],
                    [ 580.   ,    0.115],
                    [ 585.   ,    0.122],
                    [ 590.   ,    0.129],
                    [ 595.   ,    0.134],
                    [ 600.   ,    0.138],
                    [ 605.   ,    0.142],
                    [ 610.   ,    0.146],
                    [ 615.   ,    0.15 ],
                    [ 620.   ,    0.154],
                    [ 625.   ,    0.158],
                    [ 630.   ,    0.163],
                    [ 635.   ,    0.167],
                    [ 640.   ,    0.173],
                    [ 645.   ,    0.18 ],
                    [ 650.   ,    0.188],
                    [ 655.   ,    0.196],
                    [ 660.   ,    0.204],
                    [ 665.   ,    0.213],
                    [ 670.   ,    0.222],
                    [ 675.   ,    0.231],
                    [ 680.   ,    0.242],
                    [ 685.   ,    0.251],
                    [ 690.   ,    0.261],
```

(continues on next page)

(continued from previous page)

```
                    [ 695.   ,    0.271],
                    [ 700.   ,    0.282],
                    [ 705.   ,    0.294],
                    [ 710.   ,    0.305],
                    [ 715.   ,    0.318],
                    [ 720.   ,    0.334],
                    [ 725.   ,    0.354],
                    [ 730.   ,    0.372],
                    [ 735.   ,    0.392],
                    [ 740.   ,    0.409],
                    [ 745.   ,    0.42 ],
                    [ 750.   ,    0.436],
                    [ 755.   ,    0.45 ],
                    [ 760.   ,    0.462],
                    [ 765.   ,    0.465],
                    [ 770.   ,    0.448],
                    [ 775.   ,    0.432],
                    [ 780.   ,    0.421]],
                   interpolator=SpragueInterpolator,
                   interpolator_args={},
                   extrapolator=Extrapolator,
                   extrapolator_args={...})
```

colour.read_spectral_data_from_csv_file

colour.**read_spectral_data_from_csv_file**(*path*, *delimiter=', '*, *fields=None*, *default=0*)

Reads the spectral data from given *CSV* file in the following form:

390, 4.15003E-04, 3.68349E-04, 9.54729E-03 395, 1.05192E-03, 9.58658E-04, 2.38250E-02 400, 2.40836E-03, 2.26991E-03, 5.66498E-02 ... 830, 9.74306E-07, 9.53411E-08, 0.00000

and returns it as an *OrderedDict* of *dict* as follows:

OrderedDict([('field', {'wavelength': 'value', ..., 'wavelength': 'value'}), ..., ('field', {'wavelength': 'value', ..., 'wavelength': 'value'})])

Parameters

- **path** (unicode) – Absolute *CSV* file path.
- **delimiter** (unicode, optional) – *CSV* file content delimiter.
- **fields** (array_like, optional) – *CSV* file spectral data fields names. If no value is provided the first line of the file will be used as spectral data fields names.
- **default** (numeric, optional) – Default value for fields row with missing value.

Returns *CSV* file content.

Return type OrderedDict

Raises RuntimeError – If the *CSV* spectral data file doesn't define the appropriate fields.

Notes

- A *CSV* spectral data file should define at least define two fields: one for the wavelengths and one for the associated values of one spectral distribution.
- If no value is provided for the fields names, the first line of the file will be used as spectral data fields names.

Examples

```
>>> import os
>>> from pprint import pprint
>>> csv_file = os.path.join(os.path.dirname(__file__), 'tests',
...                         'resources', 'colorchecker_n_ohta.csv')
>>> sds_data = read_spectral_data_from_csv_file(csv_file)
>>> pprint(list(sds_data.keys()))
['1',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
 '9',
 '10',
 '11',
 '12',
 '13',
 '14',
 '15',
 '16',
 '17',
 '18',
 '19',
 '20',
 '21',
 '22',
 '23',
 '24']
```

colour.write_sds_to_csv_file

colour.**write_sds_to_csv_file**(*sds*, *path*, *delimiter=', '*, *fields=None*)

Writes the given spectral distributions to given *CSV* file.

Parameters

- **sds** (dict) – Spectral distributions to write.
- **path** (unicode) – Absolute *CSV* file path.
- **delimiter** (unicode, optional) – *CSV* file content delimiter.
- **fields** (array_like, optional) – *CSV* file spectral data fields names. If no value is provided the order of fields will be the one defined by the sorted spectral distributions *dict*.

Returns Definition success.

Return type bool

Raises RuntimeError – If the given spectral distributions have different shapes.

IES TM-27-14 Data

colour

SpectralDistribution_IESTM2714([path, ...])	Defines a *IES TM-27-14* spectral distribution.

colour.SpectralDistribution_IESTM2714

class colour.**SpectralDistribution_IESTM2714**(*path=None*, *header=None*, *spectral_quantity=None*, *reflection_geometry=None*, *transmission_geometry=None*, *bandwidth_FWHM=None*, *bandwidth_corrected=None*)

Bases: colour.colorimetry.spectrum.SpectralDistribution

Defines a *IES TM-27-14* spectral distribution.

This class can read and write *IES TM-27-14* spectral data XML files.

Parameters

- **path** (unicode, optional) – Spectral data XML file path.
- **header** (IES_TM2714_Header, optional) – *IES TM-27-14* spectral distribution header.
- **spectral_quantity** (unicode, optional) – **{'flux', 'absorptance', 'transmittance', 'reflectance', 'intensity', 'irradiance', 'radiance', 'exitance', 'R-Factor', 'T-Factor', 'relative', 'other'}**, Quantity of measurement for each element of the spectral data.
- **reflection_geometry** (unicode, optional) – **{'di:8', 'de:8', '8:di', '8:de', 'd:d', 'd:0', '45a:0', '45c:0', '0:45a', '45x:0', '0:45x', 'other'}**, Spectral reflectance factors geometric conditions.
- **transmission_geometry** (unicode, optional) – **{'0:0', 'di:0', 'de:0', '0:di', '0:de', 'd:d', 'other'}**, Spectral transmittance factors geometric conditions.
- **bandwidth_FWHM** (numeric, optional) – Spectroradiometer full-width half-maximum bandwidth in nanometers.
- **bandwidth_corrected** (bool, optional) – Specifies if bandwidth correction has been applied to the measured data.

Notes

Reflection Geometry

- di:8: Diffuse / eight-degree, specular component included.
- de:8: Diffuse / eight-degree, specular component excluded.
- 8:di: Eight-degree / diffuse, specular component included.
- 8:de: Eight-degree / diffuse, specular component excluded.
- d:d: Diffuse / diffuse.
- d:0: Alternative diffuse.
- 45a:0: Forty-five degree annular / normal.
- 45c:0: Forty-five degree circumferential / normal.
- 0:45a: Normal / forty-five degree annular.

- 45x:0: Forty-five degree directional / normal.
- 0:45x: Normal / forty-five degree directional.
- other: User-specified in comments.

Transmission Geometry

- 0:0: Normal / normal.
- di:0: Diffuse / normal, regular component included.
- de:0: Diffuse / normal, regular component excluded.
- 0:di: Normal / diffuse, regular component included.
- 0:de: Normal / diffuse, regular component excluded.
- d:d: Diffuse / diffuse.
- other: User-specified in comments.

mapping

path

header

spectral_quantity

reflection_geometry

transmission_geometry

bandwidth_FWHM

bandwidth_corrected

read()

write()

References

[IESCCommitteeTM2714WGroup14]

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx')).read()
>>> sd.header.manufacturer
'Unknown'
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> sd[501.7]
0.0950000...
```

property bandwidth_FWHM

Getter and setter property for the full-width half-maximum bandwidth.

Parameters **value** (numeric) – Value to set the full-width half-maximum bandwidth with.

Returns Full-width half-maximum bandwidth.

Return type numeric

property bandwidth_corrected
Getter and setter property for whether bandwidth correction has been applied to the measured data.

Parameters **value** (bool) – Whether bandwidth correction has been applied to the measured data.

Returns Whether bandwidth correction has been applied to the measured data.

Return type bool

property header
Getter and setter property for the header.

Parameters **value** (IES_TM2714_Header) – Value to set the header with.

Returns Header.

Return type IES_TM2714_Header

property mapping
Getter and setter property for the mapping structure.

Parameters **value** (Structure) – Value to set the mapping structure with.

Returns Mapping structure.

Return type *Structure*

property path
Getter and setter property for the path.

Parameters **value** (unicode) – Value to set the path with.

Returns Path.

Return type unicode

read()
Reads and parses the spectral data XML file path.

Returns Definition success.

Return type bool

Examples

```
>>> from os.path import dirname, join
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx')).read()
>>> sd.header.description
'Rare earth fluorescent lamp'
>>> # Doctests ellipsis for Python 2.x compatibility.
>>> sd[400]
0.0339999...
```

property reflection_geometry
Getter and setter property for the reflection geometry.

Parameters **value** (unicode) – Value to set the reflection geometry with.

Returns Reflection geometry.

Return type unicode

property spectral_quantity
Getter and setter property for the spectral quantity.

Parameters **value** (unicode) – Value to set the spectral quantity with.

Returns Spectral quantity.

Return type unicode

property transmission_geometry

Getter and setter property for the transmission geometry.

Parameters **value** (unicode) – Value to set the transmission geometry with.

Returns Transmission geometry.

Return type unicode

write()

Write the spectral distribution spectral data to XML file path.

Returns Definition success.

Return type bool

Examples

```
>>> from os.path import dirname, join
>>> from shutil import rmtree
>>> from tempfile import mkdtemp
>>> directory = join(dirname(__file__), 'tests', 'resources')
>>> sd = SpectralDistribution_IESTM2714(
...     join(directory, 'Fluorescent.spdx')).read()
>>> temporary_directory = mkdtemp()
>>> sd.path = join(temporary_directory, 'Fluorescent.spdx')
>>> sd.write()
True
>>> rmtree(temporary_directory)
```

X-Rite Data

colour

read_sds_from_xrite_file(path)	Reads the spectral data from given *X-Rite* file and returns it as an *OrderedDict* of colour.SpectralDistribution classes.

colour.read_sds_from_xrite_file

colour.**read_sds_from_xrite_file**(*path*)

Reads the spectral data from given *X-Rite* file and returns it as an *OrderedDict* of colour.SpectralDistribution classes.

Parameters **path** (unicode) – Absolute *X-Rite* file path.

Returns colour.SpectralDistribution classes of given *X-Rite* file.

Return type OrderedDict

Notes

- This parser is minimalistic and absolutely not bullet proof.

Examples

```
>>> import os
>>> from pprint import pprint
>>> xrite_file = os.path.join(os.path.dirname(__file__), 'tests',
...                           'resources',
...                           'X-Rite_Digital_Colour_Checker.txt')
>>> sds_data = read_sds_from_xrite_file(xrite_file)
>>> pprint(list(sds_data.keys()))
['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10']
```

Colour Models

- *Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates*
- *Common Models*
- *CIE L*a*b* Colourspace*
- *CIE L*u*v* Colourspace*
- *CIE 1960 UCS Colourspace*
- *CIE 1964 U*V*W* Colourspace*
- *Hunter L,a,b Colour Scale*
- *Hunter Rd,a,b Colour Scale*
- *DIN99 Colourspace*
- *CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)*
- *CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)*
- *IPT Colourspace*
- *hdr-CIELAB Colourspace*
- *hdr-IPT Colourspace*
- *OSA UCS Colourspace*
- *$JzAzBz$ Colourspace*
- *RGB Colourspace and Transformations*
  - *RGB Colourspace Derivation*
  - *RGB Colourspaces*
  - *Colour Component Transfer Functions*
  - *Opto-Electronic Transfer Functions*
  - *Electro-Optical Transfer Functions*
  - *Opto-Optical Transfer Functions*
  - *Log Encoding and Decoding*
  - *ACES Spectral Conversion*

- *Colour Encodings*
  - *Y'CbCr Colour Encoding*
  - *YCoCg Colour Encoding*
  - *IC_TC_P Colour Encoding*
- *RGB Representations*
  - *Prismatic Colourspace*
  - *HSV Colourspace*
  - *HSL Colourspace*
  - *CMY Colourspace*
- *Pointer's Gamut*

Tristimulus Values, CIE xyY Colourspace and Chromaticity Coordinates

colour

XYZ_to_xyY(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE xyY* colourspace and reference *illuminant*.
xyY_to_XYZ(xyY)	Converts from *CIE xyY* colourspace to *CIE XYZ* tristimulus values.
XYZ_to_xy(XYZ[, illuminant])	Returns the *CIE xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.
xy_to_XYZ(xy)	Returns the *CIE XYZ* tristimulus values from given *CIE xy* chromaticity coordinates.
xyY_to_xy(xyY)	Converts from *CIE xyY* colourspace to *CIE xy* chromaticity coordinates.
xy_to_xyY(xy[, Y])	Converts from *CIE xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with given Y *luminance*.

colour.XYZ_to_xyY

colour.**XYZ_to_xyY**(*XYZ*, *illuminant=array([0.3127, 0.329])*)

Converts from *CIE XYZ* tristimulus values to *CIE xyY* colourspace and reference *illuminant*.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant* chromaticity coordinates.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Lin03b], [Wik05a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xyY(XYZ)
array([ 0.5436955...,  0.3210794...,  0.1219722...])
```

colour.xyY_to_XYZ

colour.**xyY_to_XYZ**(*xyY*)

Converts from *CIE xyY* colourspace to *CIE XYZ* tristimulus values.

Parameters **xyY** (array_like) – *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Lin09d], [Wik05a]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_XYZ(xyY)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.XYZ_to_xy

colour.**XYZ_to_xy**(*XYZ*, *illuminant=array([0.3127, 0.329])*)

Returns the *CIE xy* chromaticity coordinates from given *CIE XYZ* tristimulus values.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik05a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_xy(XYZ)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_XYZ

colour.**xy_to_XYZ**(*xy*)

Returns the *CIE XYZ* tristimulus values from given *CIE xy* chromaticity coordinates.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xy	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik05a]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_XYZ(xy)
array([ 1.6933366...,  1.        ,  0.4211574...])
```

colour.xyY_to_xy

colour.**xyY_to_xy**(*xyY*)

Converts from *CIE xyY* colourspace to *CIE xy* chromaticity coordinates.

xyY argument with last dimension being equal to 2 will be assumed to be a *CIE xy* chromaticity coordinates argument and will be returned directly by the definition.

Parameters **xyY** (array_like) – *CIE xyY* colourspace array or *CIE xy* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Wik05a]

Examples

```
>>> xyY = np.array([0.54369557, 0.32107944, 0.12197225])
>>> xyY_to_xy(xyY)
array([ 0.54369557...,  0.32107944...])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xyY_to_xy(xy)
array([ 0.54369557...,  0.32107944...])
```

colour.xy_to_xyY

colour.**xy_to_xyY**(*xy*, *Y=1*)

Converts from *CIE xy* chromaticity coordinates to *CIE xyY* colourspace by extending the array last dimension with given *Y luminance*.

`xy` argument with last dimension being equal to 3 will be assumed to be a *CIE xyY* colourspace array argument and will be returned directly by the definition.

Parameters

- **xy** (`array_like`) – *CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (`numeric, optional`) – Optional *Y luminance* value used to construct the *CIE xyY* colourspace array, the default *Y luminance* value is 1.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
xy	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

- This definition is a convenient object provided to implement support of illuminant argument *luminance* value in various `colour.models` package objects such as `colour.Lab_to_XYZ()` or `colour.Luv_to_XYZ()`.

References

[Wik05a]

Examples

```
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy)
array([ 0.5436955...,  0.3210794...,  1.        ])
>>> xy = np.array([0.54369557, 0.32107944, 1.00000000])
>>> xy_to_xyY(xy)
array([ 0.5436955...,  0.3210794...,  1.        ])
>>> xy = np.array([0.54369557, 0.32107944])
>>> xy_to_xyY(xy, 100)
array([   0.5436955...,    0.3210794...,  100.        ])
```

Common Models

colour.models

Jab_to_JCh(Jab)	Converts from *Jab** colour representation to *JCh* colour representation.
JCh_to_Jab(JCh)	Converts from *JCh* colour representation to *Jab** colour representation.

colour.models.Jab_to_JCh

colour.models.**Jab_to_JCh**(*Jab*)

Converts from *Jab** colour representation to *JCh* colour representation.

This definition is used to perform conversion from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace and for other similar conversions. It implements a generic transformation from Lightness J, a and b opponent colour dimensions to the correlates of Lightness J, chroma C and hue angle h.

Parameters **Jab** (array_like) – *Jab** colour representation array.

Returns *JCh* colour representation array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jab	J : [0, 100] a : [-100, 100] b : [-100, 100]	J : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
JCh	J : [0, 100] C : [0, 100] h : [0, 360]	J : [0, 1] C : [0, 1] h : [0, 1]

References

[CIET14804f]

Examples

```
>>> Jab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Jab_to_JCh(Jab)
array([ 41.5278752...,  59.1242590...,  27.0884878...])
```

colour.models.JCh_to_Jab

colour.models.**JCh_to_Jab**(*JCh*)

Converts from *JCh* colour representation to *Jab** colour representation.

This definition is used to perform conversion from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace and for other similar conversions. It implements a generic transformation from the correlates of Lightness J, chroma C and hue angle h to Lightness J, a and b opponent colour dimensions.

Parameters **JCh** (array_like) – *JCh* colour representation array.

Returns *Jab** colour representation array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JCh	J : [0, 100] C : [0, 100] h : [0, 360]	J : [0, 1] C : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jab	J : [0, 100] a : [-100, 100] b : [-100, 100]	J : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804f]

Examples

```
>>> JCh = np.array([41.52787529, 59.12425901, 27.08848784])
>>> JCh_to_Jab(JCh)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CIE L*a*b* Colourspace

colour

XYZ_to_Lab(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.
Lab_to_XYZ(Lab[, illuminant])	Converts from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.
Lab_to_LCHab(Lab)	Converts from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace.
LCHab_to_Lab(LCHab)	Converts from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace.

colour.XYZ_to_Lab

colour.**XYZ_to_Lab**(*XYZ*, *illuminant=array([0.3127, 0.329])*)

Converts from *CIE XYZ* tristimulus values to *CIE L*a*b** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804f]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Lab(XYZ)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

colour.Lab_to_XYZ

colour.**Lab_to_XYZ**(*Lab*, *illuminant=array([0.3127, 0.329])*)

Converts from *CIE L*a*b** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (array_like) – *CIE L*a*b** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_XYZ(Lab)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.Lab_to_LCHab

colour.**Lab_to_LCHab**(*Lab*)

Converts from *CIE L*a*b** colourspace to *CIE L*C*Hab* colourspace.

Parameters **Lab** (array_like) – *CIE L*a*b** colourspace array.

Returns *CIE L*C*Hab* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
LCHab	L : [0, 100] C : [0, 100] Hab : [0, 360]	L : [0, 1] C : [0, 1] Hab : [0, 1]

References

[CIET14804f]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_LCHab(Lab)
array([ 41.5278752...,  59.1242590...,  27.0884878...])
```

colour.LCHab_to_Lab

colour.**LCHab_to_Lab**(*LCHab*)

Converts from *CIE L*C*Hab* colourspace to *CIE L*a*b** colourspace.

Parameters **LCHab** (array_like) – *CIE L*C*Hab* colourspace array.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
LCHab	L : [0, 100] C : [0, 100] Hab : [0, 360]	L : [0, 1] C : [0, 1] Hab : [0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[CIET14804f]

Examples

```
>>> import numpy as np
>>> LCHab = np.array([41.52787529, 59.12425901, 27.08848784])
>>> LCHab_to_Lab(LCHab)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CIE L*u*v* Colourspace

colour

XYZ_to_Luv(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE L*u*v** colourspace.
Luv_to_XYZ(Luv[, illuminant])	Converts from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.
Luv_to_LCHuv(Luv)	Converts from *CIE L*u*v** colourspace to *CIE L*C*Huv* colourspace.
LCHuv_to_Luv(LCHuv)	Converts from *CIE L*C*Huv* colourspace to *CIE L*u*v** colourspace.
Luv_to_uv(Luv[, illuminant])	Returns the uv^p chromaticity coordinates from given *CIE L*u*v** colourspace array.
uv_to_Luv(uv[, illuminant, Y])	Returns the *CIE L*u*v** colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given *L Lightness*.
Luv_uv_to_xy(uv)	Returns the *CIE xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.
xy_to_Luv_uv(xy)	Returns the *CIE L*u*v** colourspace uv^p chromaticity coordinates from given *CIE xy* chromaticity coordinates.

colour.XYZ_to_Luv

colour.**XYZ_to_Luv**(*XYZ*, *illuminant=array([0.3127, 0.329])*)

Converts from *CIE XYZ* tristimulus values to *CIE L*u*v** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE L*u*v** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

References

[CIET14804f], [Wik07c]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_Luv(XYZ)
array([ 41.5278752...,  96.8362605...,  17.7521014...])
```

colour.Luv_to_XYZ

colour.**Luv_to_XYZ**(*Luv*, *illuminant=array([0.3127, 0.329])*)

Converts from *CIE L*u*v** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Luv** (array_like) – *CIE L*u*v** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[CIET14804f], [Wik07c]

Examples

```
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_XYZ(Luv)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.Luv_to_LCHuv

colour.**Luv_to_LCHuv**(*Luv*)

Converts from *CIE L*u*v** colourspace to *CIE L*C*Huv* colourspace.

Parameters **Luv** (array_like) – *CIE L*u*v** colourspace array.

Returns *CIE L*C*Huv* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

Range	Scale - Reference	Scale - 1
LCHuv	L : [0, 100] C : [0, 100] Huv : [0, 360]	L : [0, 1] C : [0, 1] Huv : [0, 1]

References

[CIET14804f]

Examples

```
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_LCHuv(Luv)
array([ 41.5278752...,  98.4499795...,  10.3881634...])
```

colour.LCHuv_to_Luv

colour.**LCHuv_to_Luv**(*LCHuv*)

Converts from *CIE L*C*Huv* colourspace to *CIE L*u*v** colourspace.

Parameters **LCHuv** (array_like) – *CIE L*C*Huv* colourspace array.

Returns *CIE L*u*v** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
LCHuv	L : [0, 100] C : [0, 100] Huv : [0, 360]	L : [0, 1] C : [0, 1] Huv : [0, 1]

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]

References

[CIET14804f]

Examples

```
>>> LCHuv = np.array([41.52787529, 98.44997950, 10.38816348])
>>> LCHuv_to_Luv(LCHuv)
array([ 41.5278752...,  96.8362605...,  17.7521014...])
```

colour.Luv_to_uv

colour.**Luv_to_uv**(*Luv*, *illuminant=array([0.3127, 0.329])*)

Returns the uv^p chromaticity coordinates from given *CIE L*u*v** colourspace array.

Parameters

- **Luv** (array_like) – *CIE L*u*v** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns uv^p chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

References

[CIET14804e]

Examples

```
>>> Luv = np.array([41.52787529, 96.83626054, 17.75210149])
>>> Luv_to_uv(Luv)
array([ 0.3772021...,  0.5012026...])
```

colour.uv_to_Luv

colour.**uv_to_Luv**(*uv*, *illuminant=array([0.3127, 0.329])*, *Y=1*)

Returns the *CIE L*u*v** colourspace array from given uv^p chromaticity coordinates by extending the array last dimension with given L *Lightness*.

Parameters

- **uv** (array_like) – uv^p chromaticity coordinates.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y** (numeric, optional) – Optional Y *luminance* value used to construct the intermediate *CIE XYZ* colourspace array, the default Y *luminance* value is 1.

Returns *CIE L*u*v** colourspace array.

Return type ndarray

Notes

Range	Scale - Reference	Scale - 1
Luv	L : [0, 100] u : [-100, 100] v : [-100, 100]	L : [0, 1] u : [-1, 1] v : [-1, 1]
illuminant	[0, 1]	[0, 1]

References

[CIET14804e]

Examples

```
>>> uv = np.array([0.37720213, 0.50120264])
>>> uv_to_Luv(uv)
array([ 100.        ,  233.1837603...,   42.7474385...])
```

colour.Luv_uv_to_xy

colour.**Luv_uv_to_xy**(*uv*)

Returns the *CIE xy* chromaticity coordinates from given *CIE L*u*v** colourspace uv^p chromaticity coordinates.

Parameters **uv** (array_like) – *CIE L*u*v* u"v"* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

References

[Wik07e]

Examples

```
>>> uv = np.array([0.37720213, 0.50120264])
>>> Luv_uv_to_xy(uv)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_Luv_uv

colour.**xy_to_Luv_uv**(*xy*)

Returns the *CIE L*u*v** colourspace uv^p chromaticity coordinates from given *CIE xy* chromaticity coordinates.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns *CIE L*u*v* u"v"* chromaticity coordinates.

Return type ndarray

References

[Wik07c]

Examples

```
>>> xy = np.array([0.54369558, 0.32107944])
>>> xy_to_Luv_uv(xy)
array([ 0.3772021...,  0.5012026...])
```

CIE 1960 UCS Colourspace

colour

XYZ_to_UCS(XYZ)	Converts from *CIE XYZ* tristimulus values to *CIE 1960 UCS* colourspace.
UCS_to_XYZ(UVW)	Converts from *CIE 1960 UCS* colourspace to *CIE XYZ* tristimulus values.
UCS_to_uv(UVW)	Returns the *uv* chromaticity coordinates from given *CIE 1960 UCS* colourspace array.

Continued on next page

Table 151 – continued from previous page

uv_to_UCS(uv[, V])	Returns the *CIE 1960 UCS* colourspace array from given *uv* chromaticity coordinates.
UCS_uv_to_xy(uv)	Returns the *CIE xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.
xy_to_UCS_uv(xy)	Returns the *CIE 1960 UCS* colourspace *uv* chromaticity coordinates from given *CIE xy* chromaticity coordinates.

colour.XYZ_to_UCS

colour.**XYZ_to_UCS**(*XYZ*)

Converts from *CIE XYZ* tristimulus values to *CIE 1960 UCS* colourspace.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values.

Returns *CIE 1960 UCS* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

References

[Wik08d], [Wik08a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_UCS(XYZ)
array([ 0.1376933...,  0.1219722...,  0.1053731...])
```

colour.UCS_to_XYZ

colour.**UCS_to_XYZ**(*UVW*)

Converts from *CIE 1960 UCS* colourspace to *CIE XYZ* tristimulus values.

Parameters **UVW** (array_like) – *CIE 1960 UCS* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik08d], [Wik08a]

Examples

```
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_XYZ(UVW)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.UCS_to_uv

colour.**UCS_to_uv**(*UVW*)

Returns the *uv* chromaticity coordinates from given *CIE 1960 UCS* colourspace array.

Parameters **UVW** (array_like) – *CIE 1960 UCS* colourspace array.

Returns *uv* chromaticity coordinates.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
UVW	[0, 1]	[0, 1]

References

[Wik08d]

Examples

```
>>> UVW = np.array([0.13769339, 0.12197225, 0.10537310])
>>> UCS_to_uv(UVW)
array([ 0.3772021...,  0.3341350...])
```

colour.uv_to_UCS

colour.**uv_to_UCS**(*uv*, *V=1*)

Returns the *CIE 1960 UCS* colourspace array from given *uv* chromaticity coordinates.

Parameters

- **uv** (array_like) – *uv* chromaticity coordinates.
- **V** (numeric, optional) – Optional V *luminance* value used to construct the *CIE 1960 UCS* colourspace array, the default V *luminance* is set to 1.

Returns *CIE 1960 UCS* colourspace array.

Return type ndarray

References

[Wik08d]

Examples

```
>>> uv = np.array([0.37720213, 0.33413508])
>>> uv_to_UCS(uv)
array([ 1.1288911...,  1.        ,  0.8639104...])
```

colour.UCS_uv_to_xy

colour.**UCS_uv_to_xy**(*uv*)

Returns the *CIE xy* chromaticity coordinates from given *CIE 1960 UCS* colourspace *uv* chromaticity coordinates.

Parameters **uv** (array_like) – *CIE UCS uv* chromaticity coordinates.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

References

[Wik08d]

Examples

```
>>> uv = np.array([0.37720213, 0.33413508])
>>> UCS_uv_to_xy(uv)
array([ 0.5436955...,  0.3210794...])
```

colour.xy_to_UCS_uv

colour.**xy_to_UCS_uv**(*xy*)

Returns the *CIE 1960 UCS* colourspace *uv* chromaticity coordinates from given *CIE xy* chromaticity coordinates.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns *CIE UCS uv* chromaticity coordinates.

Return type ndarray

References

[Wik08d]

Examples

```
>>> xy = np.array([0.54369555, 0.32107941])
>>> xy_to_UCS_uv(xy)
array([ 0.3772021...,  0.3341350...])
```

CIE 1964 U*V*W* Colourspace

colour

XYZ_to_UVW(XYZ[, illuminant])	Converts from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.
UVW_to_XYZ(UVW[, illuminant])	Converts *CIE 1964 U*V*W** colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_UVW

colour.**XYZ_to_UVW**(*XYZ*, *illuminant=array([0.3127, 0.329])*)

Converts from *CIE XYZ* tristimulus values to *CIE 1964 U*V*W** colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE 1964 U*V*W** colourspace array.

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
UVW	U : [-100, 100] V : [-100, 100] W : [0, 100]	U : [-1, 1] V : [-1, 1] W : [0, 1]

References

[Wik08b]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_UVW(XYZ)
array([ 94.5503572...,  11.5553652...,  40.5475740...])
```

colour.UVW_to_XYZ

colour.**UVW_to_XYZ**(*UVW*, *illuminant=array([0.3127, 0.329])*)

Converts *CIE 1964 U*V*W** colourspace to *CIE XYZ* tristimulus values.

Parameters

- **UVW** (array_like) – *CIE 1964 U*V*W** colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Warning: The input domain and output range of that definition are non standard!

Notes

Domain	Scale - Reference	Scale - 1
UVW	U : [-100, 100] V : [-100, 100] W : [0, 100]	U : [-1, 1] V : [-1, 1] W : [0, 1]
illuminant	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Wik08b]

Examples

```
>>> import numpy as np
>>> UVW = np.array([94.55035725, 11.55536523, 40.54757405])
>>> UVW_to_XYZ(UVW)
array([ 20.654008,  12.197225,   5.136952])
```

Hunter L,a,b Colour Scale

colour

XYZ_to_Hunter_Lab(XYZ[, XYZ_n, K_ab])	Converts from *CIE XYZ* tristimulus values to *Hunter L,a,b* colour scale.
Hunter_Lab_to_XYZ(Lab[, XYZ_n, K_ab])	Converts from *Hunter L,a,b* colour scale to *CIE XYZ* tristimulus values.
XYZ_to_K_ab_HunterLab1966(XYZ)	Converts from *whitepoint CIE XYZ* tristimulus values to *Hunter L,a,b* K_a and K_b chromaticity coefficients.

colour.XYZ_to_Hunter_Lab

colour.**XYZ_to_Hunter_Lab**(*XYZ*, *XYZ_n=array([95.02, 100., 108.82])*, *K_ab=array([172.3, 67.2])*)

Converts from *CIE XYZ* tristimulus values to *Hunter L,a,b* colour scale.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.
- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if K_ab is set to *None* it will be computed using colour.XYZ_to_K_ab_HunterLab1966().

Returns *Hunter L,a,b* colour scale array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[Hun08a]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> XYZ_to_Hunter_Lab(XYZ, D65.XYZ_n, D65.K_ab)
array([ 34.9245257...,  47.0618985...,  14.3861510...])
```

colour.Hunter_Lab_to_XYZ

colour.**Hunter_Lab_to_XYZ**(*Lab*, *XYZ_n=array([95.02, 100., 108.82])*, *K_ab=array([172.3, 67.2])*)

Converts from *Hunter L,a,b* colour scale to *CIE XYZ* tristimulus values.

Parameters

- **Lab** (array_like) – *Hunter L,a,b* colour scale array.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.
- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if K_ab is set to *None* it will be computed using colour.XYZ_to_K_ab_HunterLab1966().

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[Hun08a]

Examples

```
>>> Lab = np.array([34.92452577, 47.06189858, 14.38615107])
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> Hunter_Lab_to_XYZ(Lab, D65.XYZ_n, D65.K_ab)
array([ 20.654008,  12.197225,   5.136952])
```

colour.XYZ_to_K_ab_HunterLab1966

colour.**XYZ_to_K_ab_HunterLab1966**(*XYZ*)

Converts from *whitepoint CIE XYZ* tristimulus values to *Hunter L,a,b* K_a and K_b chromaticity coefficients.

Parameters **XYZ** (array_like) – *Whitepoint CIE XYZ* tristimulus values.

Returns *Hunter L,a,b* K_a and K_b chromaticity coefficients.

Return type ndarray

References

[Hun08b]

Examples

```
>>> XYZ = np.array([109.850, 100.000, 35.585])
>>> XYZ_to_K_ab_HunterLab1966(XYZ)
array([ 185.2378721...,   38.4219142...])
```

Hunter Rd,a,b Colour Scale

colour

XYZ_to_Hunter_Rdab(XYZ[, XYZ_n, K_ab])	Converts from *CIE XYZ* tristimulus values to *Hunter Rd,a,b* colour scale.
Hunter_Rdab_to_XYZ(R_d_ab[, XYZ_n, K_ab])	Converts from *Hunter Rd,a,b* colour scale to *CIE XYZ* tristimulus values.

colour.XYZ_to_Hunter_Rdab

colour.**XYZ_to_Hunter_Rdab**(*XYZ*, *XYZ_n=array([95.02, 100., 108.82])*, *K_ab=array([172.3, 67.2])*)

Converts from *CIE XYZ* tristimulus values to *Hunter Rd,a,b* colour scale.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.
- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if K_ab is set to *None* it will be computed using colour.XYZ_to_K_ab_HunterLab1966().

Returns *Hunter Rd,a,b* colour scale array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
R_d_ab	R_d : [0, 100] a_Rd : [-100, 100] b_Rd : [-100, 100]	R_d : [0, 1] a_Rd : [-1, 1] b_Rd : [-1, 1]

References

[Hun12]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> XYZ_to_Hunter_Rdab(XYZ, D65.XYZ_n, D65.K_ab)
...
array([ 12.197225 ...,  57.1253787...,  17.4624134...])
```

colour.Hunter_Rdab_to_XYZ

colour.**Hunter_Rdab_to_XYZ**(*R_d_ab*, *XYZ_n=array([95.02, 100., 108.82])*, *K_ab=array([172.3, 67.2])*)

Converts from *Hunter Rd,a,b* colour scale to *CIE XYZ* tristimulus values.

Parameters

- **R_d_ab** (array_like) – *Hunter Rd,a,b* colour scale array.
- **XYZ_n** (array_like, optional) – Reference *illuminant* tristimulus values.

- **K_ab** (array_like, optional) – Reference *illuminant* chromaticity coefficients, if K_ab is set to *None* it will be computed using colour.XYZ_to_K_ab_HunterLab1966().

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
R_d_ab	R_d : [0, 100] a_Rd : [-100, 100] b_Rd : [-100, 100]	R_d : [0, 1] a_Rd : [-1, 1] b_Rd : [-1, 1]
XYZ_n	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

References

[Hun12]

Examples

```
>>> import numpy as np
>>> R_d_ab = np.array([12.19722500, 57.12537874, 17.46241341])
>>> D65 = HUNTERLAB_ILLUMINANTS[
...     'CIE 1931 2 Degree Standard Observer']['D65']
>>> Hunter_Rdab_to_XYZ(R_d_ab, D65.XYZ_n, D65.K_ab)
array([ 20.654008,  12.197225,   5.136952])
```

DIN99 Colourspace

colour

Lab_to_DIN99(Lab[, k_E, k_CH])	Converts from *CIE L*a*b** colourspace to *DIN99* colourspace.
DIN99_to_Lab(Lab_99[, k_E, k_CH])	Converts from *DIN99* colourspace to *CIE L*a*b** colourspace.

colour.Lab_to_DIN99

colour.**Lab_to_DIN99**(*Lab*, *k_E=1*, *k_CH=1*)

Converts from *CIE L*a*b** colourspace to *DIN99* colourspace.

Parameters

- **Lab** (array_like) – *CIE L*a*b** colourspace array.
- **k_E** (numeric, optional) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (numeric, optional) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.

Returns *DIN99* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

Range	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab = np.array([41.52787529, 52.63858304, 26.92317922])
>>> Lab_to_DIN99(Lab)
array([ 53.2282198...,  28.4163465...,   3.8983955...])
```

colour.DIN99_to_Lab

colour.**DIN99_to_Lab**(*Lab_99*, *k_E=1*, *k_CH=1*)

Converts from *DIN99* colourspace to *CIE L*a*b** colourspace.

Parameters

- **Lab_99** (array_like) – *DIN99* colourspace array.
- **k_E** (numeric, optional) – Parametric factor K_E used to compensate for texture and other specimen presentation effects.
- **k_CH** (numeric, optional) – Parametric factor K_{CH} used to compensate for texture and other specimen presentation effects.

Returns *CIE L*a*b** colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_99	L_99 : [0, 100] a_99 : [-100, 100] b_99 : [-100, 100]	L_99 : [0, 1] a_99 : [-1, 1] b_99 : [-1, 1]

Range	Scale - Reference	Scale - 1
Lab	L : [0, 100] a : [-100, 100] b : [-100, 100]	L : [0, 1] a : [-1, 1] b : [-1, 1]

References

[ASTMInternational07]

Examples

```
>>> import numpy as np
>>> Lab_99 = np.array([53.22821988, 28.41634656, 3.89839552])
>>> DIN99_to_Lab(Lab_99)
array([ 41.5278752...,  52.6385830...,  26.9231792...])
```

CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)

colour

JMh_CIECAM02_to_CAM02LCD(JMh)	Converts from *CIECAM02* JMh correlates array to *Luo et al.(2006) CAM02-LCD* colourspace $J'a'b'$ array..
CAM02LCD_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et al.(2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array..
JMh_CIECAM02_to_CAM02SCD(JMh)	Converts from *CIECAM02* JMh correlates array to *Luo et al.(2006) CAM02-SCD* colourspace $J'a'b'$ array..
CAM02SCD_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et al.(2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array..
JMh_CIECAM02_to_CAM02UCS(JMh)	Converts from *CIECAM02* JMh correlates array to *Luo et al.(2006) CAM02-UCS* colourspace $J'a'b'$ array..
CAM02UCS_to_JMh_CIECAM02(Jpapbp)	Converts from *Luo et al.(2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array..

colour.JMh_CIECAM02_to_CAM02LCD

colour.**JMh_CIECAM02_to_CAM02LCD**(*JMh*)

Converts from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CIECAM02* correlates array JMh.

Returns *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02LCD(JMh)
array([ 54.9043313...,  -0.0845039...,  -0.0685483...])
```

colour.CAM02LCD_to_JMh_CIECAM02

colour.**CAM02LCD_to_JMh_CIECAM02**(*Jpapbp*)

Converts from *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Luo et al. (2006) CAM02-LCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08450395, -0.06854831])
>>> CAM02LCD_to_JMh_CIECAM02(Jpapbp)
array([  4.1731091...e+01,   1.0884217...e-01,   2.1904843...e+02])
```

colour.JMh_CIECAM02_to_CAM02SCD

colour.**JMh_CIECAM02_to_CAM02SCD**(*JMh*)

Converts from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CIECAM02* correlates array JMh.

Returns *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02SCD(JMh)
array([ 54.9043313...,  -0.0843617...,  -0.0684329...])
```

colour.CAM02SCD_to_JMh_CIECAM02

colour.**CAM02SCD_to_JMh_CIECAM02**(*Jpapbp*)

Converts from *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Luo et al. (2006) CAM02-SCD* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08436178, -0.06843298])
>>> CAM02SCD_to_JMh_CIECAM02(Jpapbp)
array([  4.1731091...e+01,   1.0884217...e-01,   2.1904843...e+02])
```

colour.JMh_CIECAM02_to_CAM02UCS

colour.**JMh_CIECAM02_to_CAM02UCS**(*JMh*)

Converts from *CIECAM02* JMh correlates array to *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CIECAM02* correlates array JMh.

Returns *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LCL06]

Examples

```
>>> from colour.appearance import (
...     CIECAM02_VIEWING_CONDITIONS,
...     XYZ_to_CIECAM02)
>>> XYZ = np.array([19.01, 20.00, 21.78])
>>> XYZ_w = np.array([95.05, 100.00, 108.88])
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = XYZ_to_CIECAM02(
...     XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> JMh_CIECAM02_to_CAM02UCS(JMh)
array([ 54.9043313...,  -0.0844236...,  -0.0684831...])
```

colour.CAM02UCS_to_JMh_CIECAM02

colour.**CAM02UCS_to_JMh_CIECAM02**(*Jpapbp*)

Converts from *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array to *CIECAM02* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Luo et al. (2006) CAM02-UCS* colourspace $J'a'b'$ array.

Returns *CIECAM02* correlates array JMh.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LCL06]

Examples

```
>>> Jpapbp = np.array([54.90433134, -0.08442362, -0.06848314])
>>> CAM02UCS_to_JMh_CIECAM02(Jpapbp)
array([  4.1731091...e+01,   1.0884217...e-01,   2.1904843...e+02])
```

CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)

colour

JMh_CAM16_to_CAM16LCD(JMh, *[, coefficients])	Converts from *CAM16* JMh correlates array to *Li et al.(2017) CAM16-LCD* colourspace $J'a'b'$ array..
CAM16LCD_to_JMh_CAM16(Jpapbp, *[, coefficients])	Converts from *Li et al.(2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array..
JMh_CAM16_to_CAM16SCD(JMh, *[, coefficients])	Converts from *CAM16* JMh correlates array to *Li et al.(2017) CAM16-SCD* colourspace $J'a'b'$ array..
CAM16SCD_to_JMh_CAM16(Jpapbp, *[, coefficients])	Converts from *Li et al.(2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array..

Continued on next page

Table 157 – continued from previous page

JMh_CAM16_to_CAM16UCS(JMh, *[, coefficients])	Converts from *CAM16* JMh correlates array to *Li et al.(2017) CAM16-UCS* colourspace $J'a'b'$ array..
CAM16UCS_to_JMh_CAM16(Jpapbp, *[, coefficients])	Converts from *Li et al.(2017) CAM16-UCS* colourspace $J'a'b'$ array to *CAM16* JMh correlates array..

colour.JMh_CAM16_to_CAM16LCD

colour.**JMh_CAM16_to_CAM16LCD**(*JMh*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053)*)

Converts from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CAM16* correlates array JMh.

Returns *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to colour.JMh_CIECAM02_to_CAM02LCD() definition for an usage example.

colour.CAM16LCD_to_JMh_CAM16

colour.**CAM16LCD_to_JMh_CAM16**(*Jpapbp*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=0.77, c_1=0.007, c_2=0.0053)*)

Converts from *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Li et al. (2017) CAM16-LCD* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.CAM02LCD_to_JMh_CIECAM02()` definition for an usage example.

colour.JMh_CAM16_to_CAM16SCD

colour.**JMh_CAM16_to_CAM16SCD**(*JMh*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363)*)

Converts from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CAM16* correlates array JMh.

Returns *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.JMh_CIECAM02_to_CAM02SCD()` definition for an usage example.

colour.CAM16SCD_to_JMh_CAM16

`colour.`**`CAM16SCD_to_JMh_CAM16`**(*Jpapbp*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=1.24, c_1=0.007, c_2=0.0363)*)

Converts from *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** (`array_like`) – *Li et al. (2017) CAM16-SCD* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to `colour.CAM02SCD_to_JMh_CIECAM02()` definition for an usage example.

colour.JMh_CAM16_to_CAM16UCS

colour.**JMh_CAM16_to_CAM16UCS**(*JMh*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007, c_2=0.0228)*)

Converts from *CAM16* JMh correlates array to *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Parameters **JMh** (array_like) – *CAM16* correlates array JMh.

Returns *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

Range	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to colour.JMh_CIECAM02_to_CAM02UCS() definition for an usage example.

colour.CAM16UCS_to_JMh_CAM16

colour.**CAM16UCS_to_JMh_CAM16**(*Jpapbp*, ***, *coefficients=Coefficients_UCS_Luo2006(K_L=1.0, c_1=0.007, c_2=0.0228)*)

Converts from *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array to *CAM16* JMh correlates array.

Parameters **Jpapbp** (array_like) – *Li et al. (2017) CAM16-UCS* colourspace $J'a'b'$ array.

Returns *CAM16* correlates array JMh.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Jpapbp	Jp_1 : [0, 100] ap_1 : [-100, 100] bp_1 : [-100, 100]	Jp_1 : [0, 1] ap_1 : [-1, 1] bp_1 : [-1, 1]

Range	Scale - Reference	Scale - 1
JMh	J : [0, 100] M : [0, 100] h : [0, 360]	J : [0, 1] M : [0, 1] h : [0, 1]

References

[LLW+17]

Notes

- This docstring is automatically generated, please refer to colour.CAM02UCS_to_JMh_CIECAM02() definition for an usage example.

IPT Colourspace

colour

XYZ_to_IPT(XYZ)	Converts from *CIE XYZ* tristimulus values to *IPT* colourspace.
IPT_to_XYZ(IPT)	Converts from *IPT* colourspace to *CIE XYZ* tristimulus values.
IPT_hue_angle(IPT)	Computes the hue angle in degrees from *IPT* colourspace.

colour.XYZ_to_IPT

colour.**XYZ_to_IPT**(*XYZ*)

Converts from *CIE XYZ* tristimulus values to *IPT* colourspace.

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values.

Returns *IPT* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

- Input *CIE XYZ* tristimulus values needs to be adapted for *CIE Standard Illuminant D Series D65*.

References

[Fai13d]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_IPT(XYZ)
array([ 0.3842619...,  0.3848730...,  0.1888683...])
```

colour.IPT_to_XYZ

colour.**IPT_to_XYZ**(*IPT*)

Converts from *IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters **IPT** (array_like) – *IPT* colourspace array.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[Fai13d]

Examples

```
>>> IPT = np.array([0.38426191, 0.38487306, 0.18886838])
>>> IPT_to_XYZ(IPT)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.IPT_hue_angle

colour.**IPT_hue_angle**(*IPT*)

Computes the hue angle in degrees from *IPT* colourspace.

Parameters **IPT** (array_like) – *IPT* colourspace array.

Returns Hue angle in degrees.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT	I : [0, 1] P : [-1, 1] T : [-1, 1]	I : [0, 1] P : [-1, 1] T : [-1, 1]

Range	Scale - Reference	Scale - 1
hue	[0, 360]	[0, 1]

References

[Fai13d]

Examples

```
>>> IPT = np.array([0.96907232, 1, 1.12179215])
>>> IPT_hue_angle(IPT)
48.2852074...
```

hdr-CIELAB Colourspace

colour

XYZ_to_hdr_CIELab(XYZ[, illuminant, Y_s, ...])	Converts from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.
hdr_CIELab_to_XYZ(Lab_hdr[, illuminant, ...])	Converts from *hdr-CIELAB* colourspace to *CIE XYZ* tristimulus values.
HDR_CIELAB_METHODS	Supported *hdr-CIELAB* colourspace computation methods.

colour.XYZ_to_hdr_CIELab

colour.**XYZ_to_hdr_CIELab**(*XYZ*, *illuminant=array([0.3127, 0.329])*, *Y_s=0.2*, *Y_abs=100*, *method='Fairchild 2011'*)

Converts from *CIE XYZ* tristimulus values to *hdr-CIELAB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (unicode, optional) – **{'Fairchild 2011', 'Fairchild 2010'}**, Computation method.

Returns *hdr-CIELAB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lab_hdr	L_hdr : [0, 100] a_hdr : [-100, 100] b_hdr : [-100, 100]	L_hdr : [0, 1] a_hdr : [-1, 1] b_hdr : [-1, 1]

- Conversion to polar coordinates to compute the *chroma* C_{hdr} and *hue* h_{hdr} correlates can be safely performed with colour.Lab_to_LCHab() definition.
- Conversion to cartesian coordinates from the *Lightness* L_{hdr}, *chroma* C_{hdr} and *hue* h_{hdr} correlates can be safely performed with colour.LCHab_to_Lab() definition.

References

[FW10], [FC11]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_CIELab(XYZ)
array([ 51.8700206...,  60.4763385...,  32.1455191...])
>>> XYZ_to_hdr_CIELab(XYZ, method='Fairchild 2010')
array([  31.9962111...,  128.0076303...,   48.7695230...])
```

colour.hdr_CIELab_to_XYZ

colour.**hdr_CIELab_to_XYZ**(*Lab_hdr*, *illuminant=array([0.3127, 0.329])*, *Y_s=0.2*, *Y_abs=100*, *method='Fairchild 2011'*)

Converts from *hdr-CIELAB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **Lab_hdr** (array_like) – *hdr-CIELAB* colourspace array.
- **illuminant** (array_like, optional) – Reference *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (unicode, optional) – **{'Fairchild 2011', 'Fairchild 2010'}**, Computation method.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lab_hdr	L_hdr : [0, 100] a_hdr : [-100, 100] b_hdr : [-100, 100]	L_hdr : [0, 1] a_hdr : [-1, 1] b_hdr : [-1, 1]
illuminant	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[FW10], [FC11]

Examples

```
>>> Lab_hdr = np.array([51.87002062, 60.4763385, 32.14551912])
>>> hdr_CIELab_to_XYZ(Lab_hdr)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
>>> Lab_hdr = np.array([31.99621114, 128.00763036, 48.76952309])
>>> hdr_CIELab_to_XYZ(Lab_hdr, method='Fairchild 2010')
...
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.HDR_CIELAB_METHODS

colour.**HDR_CIELAB_METHODS = ('Fairchild 2010', 'Fairchild 2011')**
Supported *hdr-CIELAB* colourspace computation methods.

References

[FW10], [FC11]

HDR_CIELAB_METHODS [tuple] **{'Fairchild 2011', 'Fairchild 2010'}**

hdr-IPT Colourspace

colour

XYZ_to_hdr_IPT(XYZ[, Y_s, Y_abs, method])	Converts from *CIE XYZ* tristimulus values to *hdr-IPT* colourspace.
hdr_IPT_to_XYZ(IPT_hdr[, Y_s, Y_abs, method])	Converts from *hdr-IPT* colourspace to *CIE XYZ* tristimulus values.
HDR_IPT_METHODS	Supported *hdr-IPT* colourspace computation methods.

colour.XYZ_to_hdr_IPT

colour.**XYZ_to_hdr_IPT**(*XYZ*, *Y_s=0.2*, *Y_abs=100*, *method='Fairchild 2011'*)
Converts from *CIE XYZ* tristimulus values to *hdr-IPT* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (unicode, optional) – **{'Fairchild 2011', 'Fairchild 2010'}**, Computation method.

Returns *hdr-IPT* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
IPT_hdr	I_hdr : [0, 100] P_hdr : [-100, 100] T_hdr : [-100, 100]	I_hdr : [0, 1] P_hdr : [-1, 1] T_hdr : [-1, 1]

- Input *CIE XYZ* tristimulus values needs to be adapted for *CIE Standard Illuminant D Series D65*.

References

[FW10], [FC11]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_hdr_IPT(XYZ)
array([ 48.3937634...,  42.4499020...,  22.0195403...])
>>> XYZ_to_hdr_IPT(XYZ, method='Fairchild 2010')
array([ 30.0287314...,  83.9384506...,  34.9028738...])
```

colour.hdr_IPT_to_XYZ

colour.**hdr_IPT_to_XYZ**(*IPT_hdr*, *Y_s=0.2*, *Y_abs=100*, *method='Fairchild 2011'*)

Converts from *hdr-IPT* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **IPT_hdr** (array_like) – *hdr-IPT* colourspace array.
- **Y_s** (numeric or array_like) – Relative luminance Y_s of the surround.
- **Y_abs** (numeric or array_like) – Absolute luminance Y_{abs} of the scene diffuse white in cd/m^2.
- **method** (unicode, optional) – **{'Fairchild 2011', 'Fairchild 2010'}**, Computation method.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
IPT_hdr	I_hdr : [0, 100] P_hdr : [-100, 100] T_hdr : [-100, 100]	I_hdr : [0, 1] P_hdr : [-1, 1] T_hdr : [-1, 1]
Y_s	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[FW10], [FC11]

Examples

```
>>> IPT_hdr = np.array([48.39376346, 42.44990202, 22.01954033])
>>> hdr_IPT_to_XYZ(IPT_hdr)
array([ 0.2065400...,  0.1219722...,  0.0513695...])
>>> IPT_hdr = np.array([30.02873147, 83.93845061, 34.90287382])
>>> hdr_IPT_to_XYZ(IPT_hdr, method='Fairchild 2010')
...
array([ 0.2065400...,  0.1219722...,  0.0513695...])
```

colour.HDR_IPT_METHODS

colour.**HDR_IPT_METHODS = ('Fairchild 2010', 'Fairchild 2011')**

Supported *hdr-IPT* colourspace computation methods.

References

[FW10], [FC11]

HDR_IPT_METHODS [tuple] **{'Fairchild 2011', 'Fairchild 2010'}**

OSA UCS Colourspace

colour

XYZ_to_OSA_UCS(XYZ)	Converts from *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer* to *OSA UCS* colourspace.
OSA_UCS_to_XYZ(Ljg[, optimisation_parameters])	Converts from *OSA UCS* colourspace to *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

colour.XYZ_to_OSA_UCS

colour.**XYZ_to_OSA_UCS**(*XYZ*)

Converts from *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer* to *OSA UCS* colourspace.

The lightness axis, L is usually in range [-9, 5] and centered around middle gray (Munsell N/6). The yellow-blue axis, j is usually in range [-15, 15]. The red-green axis, g is usually in range [-20, 15].

Parameters **XYZ** (array_like) – *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Returns *OSA UCS* Ljg lightness, jaune (yellowness), and greenness.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
Ljg	L : [-100, 100] j : [-100, 100] g : [-100, 100]	L : [-1, 1] j : [-1, 1] g : [-1, 1]

- *OSA UCS* uses the *CIE 1964 10 Degree Standard Observer*.

References

[CTS13], [Mor03]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952]) * 100
>>> XYZ_to_OSA_UCS(XYZ)
array([-3.0049979...,  2.9971369..., -9.6678423...])
```

colour.OSA_UCS_to_XYZ

colour.**OSA_UCS_to_XYZ**(*Ljg*, *optimisation_parameters=None*)

Converts from *OSA UCS* colourspace to *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Parameters

- **Ljg** (array_like) – *OSA UCS* Ljg lightness, jaune (yellowness), and greenness.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.fmin() definition.

Returns *CIE XYZ* tristimulus values under the *CIE 1964 10 Degree Standard Observer*.

Return type ndarray

Warning: There is no analytical inverse transformation from *OSA UCS* to Ljg lightness, jaune (yellowness), and greenness to *CIE XYZ* tristimulus values, the current implementation relies on optimization using scipy.optimize.fmin() definition and thus has reduced precision and poor performance.

Notes

Domain	Scale - Reference	Scale - 1
Ljg	L : [-100, 100] j : [-100, 100] g : [-100, 100]	L : [-1, 1] j : [-1, 1] g : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- *OSA UCS* uses the *CIE 1964 10 Degree Standard Observer*.

References

[CTS13], [Mor03]

Examples

```
>>> import numpy as np
>>> Ljg = np.array([-3.00499790, 2.99713697, -9.66784231])
>>> OSA_UCS_to_XYZ(Ljg)
array([ 20.6540240...,  12.1972369...,   5.1369372...])
```

$JzAzBz$ Colourspace

colour

XYZ_to_JzAzBz(XYZ_D65[, constants])	Converts from *CIE XYZ* tristimulus values to $J_zA_zB_z$ colourspace.
JzAzBz_to_XYZ(JzAzBz[, constants])	Converts from $J_zA_zB_z$ colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_JzAzBz

colour.**XYZ_to_JzAzBz**(*XYZ_D65*, *constants={'b': 1.15, 'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'd': -0.56, 'd_0': 1.6295499532821565e-11, 'g': 0.66, 'm_1': 0.1593017578125, 'm_2': 134.03437499999998}*)

Converts from *CIE XYZ* tristimulus values to $J_zA_zB_z$ colourspace.

Parameters

- **XYZ_D65** (array_like) – *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.
- **constants** (Structure, optional) – $J_zA_zB_z$ colourspace constants.

Returns $J_zA_zB_z$ colourspace array where J_z is Lightness, A_z is redness-greenness and B_z is yellowness-blueness.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
JzAzBz	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]

References

[SCKL17]

Examples

```
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_JzAzBz(XYZ)
array([ 0.0053504...,  0.0092430...,  0.0052600...])
```

colour.JzAzBz_to_XYZ

colour.**JzAzBz_to_XYZ**(*JzAzBz*, *constants={'b': 1.15, 'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'd': -0.56, 'd_0': 1.6295499532821565e-11, 'g': 0.66, 'm_1': 0.1593017578125, 'm_2': 134.03437499999998}*)

Converts from $J_zA_zB_z$ colourspace to *CIE XYZ* tristimulus values.

Parameters

- **JzAzBz** (array_like) – $J_zA_zB_z$ colourspace array where J_z is Lightness, A_z is redness-greenness and B_z is yellowness-blueness.
- **constants** (Structure, optional) – $J_zA_zB_z$ colourspace constants.

Returns *CIE XYZ* tristimulus values under *CIE Standard Illuminant D Series D65*.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
JzAzBz	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]	Jz : [0, 1] Az : [-1, 1] Bz : [-1, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

References

[SCKL17]

Examples

```
>>> JzAzBz = np.array([0.00535048, 0.00924302, 0.00526007])
>>> JzAzBz_to_XYZ(JzAzBz)
array([ 0.2065402...,  0.1219723...,  0.0513696...])
```

RGB Colourspace and Transformations

colour

XYZ_to_RGB(XYZ, illuminant_XYZ, ... [, ...])	Converts from *CIE XYZ* tristimulus values to *RGB* colourspace array.
RGB_to_XYZ(RGB, illuminant_RGB, ... [, ...])	Converts given *RGB* colourspace array to *CIE XYZ* tristimulus values.
RGB_to_RGB(RGB, input_colourspace, ... [, ...])	Converts given *RGB* colourspace array from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.
RGB_to_RGB_matrix(input_colourspace, ... [, ...])	Computes the matrix M converting from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

colour.XYZ_to_RGB

colour.**XYZ_to_RGB**(*XYZ*, *illuminant_XYZ*, *illuminant_RGB*, *XYZ_to_RGB_matrix*, *chromatic_adaptation_transform='CAT02'*, *cctf_encoding=None*, ***kwargs*)

Converts from *CIE XYZ* tristimulus values to *RGB* colourspace array.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant_XYZ** (array_like) – *CIE XYZ* tristimulus values *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **illuminant_RGB** (array_like) – *RGB* colourspace *illuminant CIE xy* chromaticity coordinates or *CIE xyY* colourspace array.
- **XYZ_to_RGB_matrix** (array_like) – *Normalised primary matrix*.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **cctf_encoding** (object, optional) – Encoding colour component transfer function (Encoding CCTF) or opto-electronic transfer function (OETF / OECF).

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]
illuminant_XYZ	[0, 1]	[0, 1]
illuminant_RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> XYZ = np.array([0.21638819, 0.12570000, 0.03847493])
>>> illuminant_XYZ = np.array([0.34570, 0.35850])
>>> illuminant_RGB = np.array([0.31270, 0.32900])
>>> chromatic_adaptation_transform = 'Bradford'
>>> XYZ_to_RGB_matrix = np.array(
...     [[3.24062548, -1.53720797, -0.49862860],
...      [-0.96893071, 1.87575606, 0.04151752],
...      [0.05571012, -0.20402105, 1.05699594]]
... )
>>> XYZ_to_RGB(XYZ, illuminant_XYZ, illuminant_RGB, XYZ_to_RGB_matrix,
...            chromatic_adaptation_transform)
array([ 0.4559557...,  0.0303970...,  0.0408724...])
```

colour.RGB_to_XYZ

colour.**RGB_to_XYZ**(*RGB*, *illuminant_RGB*, *illuminant_XYZ*, *RGB_to_XYZ_matrix*, *chromatic_adaptation_transform='CAT02'*, *cctf_decoding=None*, ***kwargs*)

Converts given *RGB* colourspace array to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **illuminant_RGB** (array_like) – *RGB* colourspace *illuminant* chromaticity coordinates or *CIE xyY* colourspace array.
- **illuminant_XYZ** (array_like) – *CIE XYZ* tristimulus values *illuminant* chromaticity coordinates or *CIE xyY* colourspace array.
- **RGB_to_XYZ_matrix** (array_like) – *Normalised primary matrix*.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **cctf_decoding** (object, optional) – Decoding colour component transfer function (Decoding CCTF) or electro-optical transfer function (EOTF / EOCF).

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]
illuminant_XYZ	[0, 1]	[0, 1]
illuminant_RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> illuminant_RGB = np.array([0.31270, 0.32900])
>>> illuminant_XYZ = np.array([0.34570, 0.35850])
>>> chromatic_adaptation_transform = 'Bradford'
>>> RGB_to_XYZ_matrix = np.array(
...     [[0.41240000, 0.35760000, 0.18050000],
...      [0.21260000, 0.71520000, 0.07220000],
...      [0.01930000, 0.11920000, 0.95050000]]
... )
>>> RGB_to_XYZ(RGB, illuminant_RGB, illuminant_XYZ, RGB_to_XYZ_matrix,
...            chromatic_adaptation_transform)
array([ 0.2163881...,  0.1257    ,  0.0384749...])
```

colour.RGB_to_RGB

colour.**RGB_to_RGB**(*RGB*, *input_colourspace*, *output_colourspace*, *chromatic_adaptation_transform='CAT02'*, *apply_cctf_decoding=False*, *apply_cctf_encoding=False*, ***kwargs*)

Converts given *RGB* colourspace array from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **input_colourspace** (RGB_Colourspace) – *RGB* input colourspace.
- **output_colourspace** (RGB_Colourspace) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.
- **apply_cctf_decoding** (bool, optional) – Apply input colourspace decoding colour component transfer function / electro-optical transfer function.
- **apply_cctf_encoding** (bool, optional) – Apply output colourspace encoding colour component transfer function / opto-electronic transfer function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for the colour component transfer functions.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> from colour.models import sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE
>>> RGB = np.array([0.45595571, 0.03039702, 0.04087245])
>>> RGB_to_RGB(RGB, sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE)
...
array([ 0.2568891...,  0.0721446...,  0.0465553...])
```

colour.RGB_to_RGB_matrix

colour.**RGB_to_RGB_matrix**(*input_colourspace*, *output_colourspace*, *chromatic_adaptation_transform='CAT02'*)

Computes the matrix M converting from given input *RGB* colourspace to output *RGB* colourspace using given *chromatic adaptation* method.

Parameters

- **input_colourspace** (RGB_Colourspace) – *RGB* input colourspace.
- **output_colourspace** (RGB_Colourspace) – *RGB* output colourspace.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC', None}**, *Chromatic adaptation* transform, if *None* no chromatic adaptation is performed.

Returns Conversion matrix M.

Return type ndarray

Examples

```
>>> from colour.models import sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE
>>> RGB_to_RGB_matrix(sRGB_COLOURSPACE, PROPHOTO_RGB_COLOURSPACE)
...
array([[ 0.5288241...,  0.3340609...,  0.1373616...],
       [ 0.0975294...,  0.8790074...,  0.0233981...],
       [ 0.0163599...,  0.1066124...,  0.8772485...]])
```

Ancillary Objects

colour

XYZ_to_sRGB(XYZ[, illuminant, ...])	Converts from *CIE XYZ* tristimulus values to *sRGB* colourspace.
sRGB_to_XYZ(RGB[, illuminant, ...])	Converts from *sRGB* colourspace to *CIE XYZ* tristimulus values.

colour.XYZ_to_sRGB

colour.**XYZ_to_sRGB**(*XYZ*, *illuminant=array([0.3127, 0.329])*, *chromatic_adaptation_transform='CAT02'*, *apply_cctf_encoding=True*, ***kwargs*)

Converts from *CIE XYZ* tristimulus values to *sRGB* colourspace.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **illuminant** (array_like, optional) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.
- **apply_cctf_encoding** (bool, optional) – Apply *sRGB* encoding colour component transfer function / opto-electronic transfer function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *sRGB* colour array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> XYZ_to_sRGB(XYZ)
array([ 0.7057393...,  0.1924826...,  0.2235416...])
```

colour.sRGB_to_XYZ

colour.**sRGB_to_XYZ**(*RGB*, *illuminant=array([0.3127, 0.329])*, *chromatic_adaptation_method='CAT02'*, *apply_cctf_decoding=True*, ***kwargs*)

Converts from *sRGB* colourspace to *CIE XYZ* tristimulus values.

Parameters

- **RGB** (array_like) – *sRGB* colourspace array.
- **illuminant** (array_like, optional) – Source illuminant chromaticity coordinates.
- **chromatic_adaptation_method** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* method.

- **apply_cctf_decoding** (bool, optional) – Apply *sRGB* decoding colour component transfer function / electro-optical transfer function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *CIE XYZ* tristimulus values.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> RGB = np.array([0.70573936, 0.19248266, 0.22354169])
>>> sRGB_to_XYZ(RGB)
array([ 0.2065429...,  0.1219794...,  0.0513714...])
```

RGB Colourspace Derivation

colour

normalised_primary_matrix(primaries, whitepoint)	Returns the *normalised primary matrix* using given *primaries* and *whitepoint* xy chromaticity coordinates.
chromatically_adapted_primaries(primaries, ...)	Chromatically adapts given *primaries* xy chromaticity coordinates from test whitepoint_t to reference whitepoint_r.
primaries_whitepoint(npm)	Returns the *primaries* and *whitepoint* xy chromaticity coordinates using given *normalised primary matrix*.
RGB_luminance(RGB, primaries, whitepoint)	Returns the *luminance* Y of given *RGB* components from given *primaries* and *whitepoint*.
RGB_luminance_equation(primaries, whitepoint)	Returns the *luminance equation* from given *primaries* and *whitepoint*.

colour.normalised_primary_matrix

colour.**normalised_primary_matrix**(*primaries*, *whitepoint*)

Returns the *normalised primary matrix* using given *primaries* and *whitepoint* xy chromaticity coordinates.

Parameters

- **primaries** (array_like, (3, 2)) – Primaries xy chromaticity coordinates.
- **whitepoint** (array_like) – Illuminant / whitepoint xy chromaticity coordinates.

Returns *Normalised primary matrix*.

Return type ndarray, (3, 3)

References

[SocietyoMPaTEngineers93]

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> w = np.array([0.32168, 0.33767])
>>> normalised_primary_matrix(p, w)
array([[  9.5255239...e-01,   0.0000000...e+00,   9.3678631...e-05],
       [  3.4396645...e-01,   7.2816609...e-01,  -7.2132546...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   1.0088251...e+00]])
```

colour.chromatically_adapted_primaries

colour.**chromatically_adapted_primaries**(*primaries*, *whitepoint_t*, *whitepoint_r*, *chromatic_adaptation_transform='CAT02'*)

Chromatically adapts given *primaries* xy chromaticity coordinates from test whitepoint_t to reference whitepoint_r.

Parameters

- **primaries** (array_like, (3, 2)) – Primaries xy chromaticity coordinates.
- **whitepoint_t** (array_like) – Test illuminant / whitepoint xy chromaticity coordinates.
- **whitepoint_r** (array_like) – Reference illuminant / whitepoint xy chromaticity coordinates.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.

Returns Chromatically adapted primaries xy chromaticity coordinates.

Return type ndarray

Examples

```
>>> p = np.array([0.64, 0.33, 0.30, 0.60, 0.15, 0.06])
>>> w_t = np.array([0.31270, 0.32900])
>>> w_r = np.array([0.34570, 0.35850])
>>> chromatic_adaptation_transform = 'Bradford'
>>> chromatically_adapted_primaries(p, w_t, w_r,
...                                 chromatic_adaptation_transform)
...
array([[ 0.6484414...,  0.3308533...],
       [ 0.3211951...,  0.5978443...],
       [ 0.1558932...,  0.0660492...]])
```

colour.primaries_whitepoint

colour.**primaries_whitepoint**(*npm*)

Returns the *primaries* and *whitepoint* xy chromaticity coordinates using given *normalised primary matrix*.

Parameters **npm** (array_like, (3, 3)) – *Normalised primary matrix*.

Returns *Primaries* and *whitepoint* xy chromaticity coordinates.

Return type tuple

References

[Tri15]

Examples

```
>>> npm = np.array([[9.52552396e-01, 0.00000000e+00, 9.36786317e-05],
...                 [3.43966450e-01, 7.28166097e-01, -7.21325464e-02],
...                 [0.00000000e+00, 0.00000000e+00, 1.00882518e+00]])
>>> p, w = primaries_whitepoint(npm)
>>> p
array([[  7.3470000...e-01,   2.6530000...e-01],
       [  0.0000000...e+00,   1.0000000...e+00],
       [  1.0000000...e-04,  -7.7000000...e-02]])
>>> w
array([ 0.32168,  0.33767])
```

colour.RGB_luminance

colour.**RGB_luminance**(*RGB*, *primaries*, *whitepoint*)

Returns the *luminance* Y of given *RGB* components from given *primaries* and *whitepoint*.

Parameters

- **RGB** (array_like) – *RGB* chromaticity coordinate matrix.
- **primaries** (array_like, (3, 2)) – Primaries chromaticity coordinate matrix.
- **whitepoint** (array_like) – Illuminant / whitepoint chromaticity coordinates.

Returns *Luminance* Y.

Return type numeric or ndarray

Examples

```
>>> RGB = np.array([0.21959402, 0.06986677, 0.04703877])
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_luminance(RGB, p, whitepoint)
0.1230145...
```

colour.RGB_luminance_equation

colour.**RGB_luminance_equation**(*primaries*, *whitepoint*)

Returns the *luminance equation* from given *primaries* and *whitepoint*.

Parameters

- **primaries** (array_like, (3, 2)) – Primaries chromaticity coordinates.
- **whitepoint** (array_like) – Illuminant / whitepoint chromaticity coordinates.

Returns *Luminance* equation.

Return type unicode

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> # Doctests skip for Python 2.x compatibility.
>>> RGB_luminance_equation(p, whitepoint)
'Y = 0.3439664...(R) + 0.7281660...(G) + -0.0721325...(B)'
```

RGB Colourspaces

colour

RGB_Colourspace(name, primaries, whitepoint)	Implements support for the *RGB* colourspaces datasets from colour.models.datasets.aces_rgb, etc....

colour.RGB_Colourspace

class colour.**RGB_Colourspace**(*name*, *primaries*, *whitepoint*, *whitepoint_name=None*, *RGB_to_XYZ_matrix=None*, *XYZ_to_RGB_matrix=None*, *cctf_encoding=None*, *cctf_decoding=None*, *use_derived_RGB_to_XYZ_matrix=False*, *use_derived_XYZ_to_RGB_matrix=False*)

Bases: object

Implements support for the *RGB* colourspaces datasets from colour.models.datasets.aces_rgb, etc....

Colour science literature related to *RGB* colourspaces and encodings defines their dataset using different degree of precision or rounding. While instances where a whitepoint is being defined with a value different than its canonical agreed one are rare, it is however very common to have normalised primary matrices rounded at different decimals. This can yield large discrepancies in computations.

Such an occurrence is the *V-Gamut* colourspace white paper, that defines the *V-Gamut* to *ITU-R BT.709* conversion matrix as follows:

```
[[ 1.806576 -0.695697 -0.110879]
 [-0.170090  1.305955 -0.135865]
 [-0.025206 -0.154468  1.179674]]
```

Computing this matrix using *ITU-R BT.709* colourspace derived normalised primary matrix yields:

```
[[ 1.8065736 -0.6956981 -0.1108786]
 [-0.1700890  1.3059548 -0.1358648]
 [-0.0252057 -0.1544678  1.1796737]]
```

The latter matrix is almost equals with the former, however performing the same computation using *IEC 61966-2-1:1999 sRGB* colourspace normalised primary matrix introduces severe disparities:

```
[[ 1.8063853 -0.6956147 -0.1109453]
 [-0.1699311  1.3058387 -0.1358616]
 [-0.0251630 -0.1544899  1.1797117]]
```

In order to provide support for both literature defined dataset and accurate computations enabling transformations without loss of precision, the `colour.RGB_Colourspace` class provides two sets of transformation matrices:

- Instantiation transformation matrices
- Derived transformation matrices

Upon instantiation, the `colour.RGB_Colourspace` class stores the given `RGB_to_XYZ_matrix` and `XYZ_to_RGB_matrix` arguments and also computes their derived counterpart using the `primaries` and `whitepoint` arguments.

Whether the initialisation or derived matrices are used in subsequent computations is dependent on the `colour.RGB_Colourspace.use_derived_RGB_to_XYZ_matrix` and `colour.RGB_Colourspace.use_derived_XYZ_to_RGB_matrix` attributes values.

Parameters

- **name** (unicode) – *RGB* colourspace name.
- **primaries** (array_like) – *RGB* colourspace primaries.
- **whitepoint** (array_like) – *RGB* colourspace whitepoint.
- **whitepoint_name** (unicode, optional) – *RGB* colourspace whitepoint name.
- **RGB_to_XYZ_matrix** (array_like, optional) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **XYZ_to_RGB_matrix** (array_like, optional) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **cctf_encoding** (object, optional) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF) that maps estimated tristimulus values in a scene to $R'G'B'$ video component signal value.
- **cctf_decoding** (object, optional) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF) that maps an $R'G'B'$ video component signal value to tristimulus values at the display.
- **use_derived_RGB_to_XYZ_matrix** (bool, optional) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

- **use_derived_XYZ_to_RGB_matrix** (bool, optional) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

name

primaries

whitepoint

whitepoint_name

RGB_to_XYZ_matrix

XYZ_to_RGB_matrix

cctf_encoding

cctf_decoding

use_derived_RGB_to_XYZ_matrix

use_derived_XYZ_to_RGB_matrix

__str__()

__repr__()

use_derived_transformation_matrices()

chromatically_adapt()

copy()

Notes

- The normalised primary matrix defined by colour.RGB_Colourspace.RGB_to_XYZ_matrix attribute is treated as the prime matrix from which the inverse will be calculated as required by the internal derivation mechanism. This behaviour has been chosen in accordance with literature where commonly a *RGB* colourspace is defined by its normalised primary matrix as it is directly computed from the chosen primaries and whitepoint.

References

[InternationalECommission99], [Pan14]

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_to_XYZ_matrix = np.identity(3)
>>> XYZ_to_RGB_matrix = np.identity(3)
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                               RGB_to_XYZ_matrix, XYZ_to_RGB_matrix)
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
True
```

(continues on next page)

(continued from previous page)

```
>>> colourspace.RGB_to_XYZ_matrix
array([[  9.5255239...e-01,   0.0000000...e+00,   9.3678631...e-05],
       [  3.4396645...e-01,   7.2816609...e-01,  -7.2132546...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   1.0088251...e+00]])
>>> colourspace.XYZ_to_RGB_matrix
array([[  1.0498110...e+00,   0.0000000...e+00,  -9.7484540...e-05],
       [ -4.9590302...e-01,   1.3733130...e+00,   9.8240036...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   9.9125201...e-01]])
>>> colourspace.use_derived_RGB_to_XYZ_matrix = False
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_XYZ_to_RGB_matrix = False
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

property RGB_to_XYZ_matrix

Getter and setter property for the transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Parameters **value** (array_like) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Returns Transformation matrix from colourspace to *CIE XYZ* tristimulus values.

Return type array_like

property XYZ_to_RGB_matrix

Getter and setter property for the transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Parameters **value** (array_like) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Returns Transformation matrix from *CIE XYZ* tristimulus values to colourspace.

Return type array_like

property cctf_decoding

Getter and setter property for the decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).

Parameters **value** (callable) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).

Returns Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF).

Return type callable

property cctf_encoding

Getter and setter property for the encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).

Parameters **value** (callable) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).

Returns Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF).

Return type callable

chromatically_adapt(*whitepoint*, *whitepoint_name=None*, *chromatic_adaptation_transform='CAT02'*)

Chromatically adapts the *RGB* colourspace *primaries* xy chromaticity coordinates from *RGB* colourspace whitepoint to reference `whitepoint`.

Parameters

- **whitepoint** (`array_like`) – Reference illuminant / whitepoint xy chromaticity coordinates.
- **whitepoint_name** (`unicode, optional`) – Reference illuminant / whitepoint name.
- **chromatic_adaptation_transform** (`unicode, optional`) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.

Returns

Return type Chromatically adapted *RGB* colourspace.

Examples

```
>>> p = np.array(
...     [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> w_t = np.array([0.32168, 0.33767])
>>> w_r = np.array([0.31270, 0.32900])
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, w_t, 'D65')
>>> print(colourspace.chromatically_adapt(w_r, 'D50', 'Bradford'))
...
RGB Colourspace - Chromatically Adapted to [ 0.3127  0.329 ]
------------------------------------------------------------

Primaries          : [[ 0.73485524  0.26422533]
                      [-0.00617091  1.01131496]
                      [ 0.01596756 -0.0642355 ]]
Whitepoint         : [ 0.3127  0.329 ]
Whitepoint Name    : D50
Encoding CCTF      : None
Decoding CCTF      : None
NPM                : None
NPM -1             : None
Derived NPM        : [[ 0.93827985 -0.00445145  0.01662752]
                      [ 0.33736889  0.72952157 -0.06689046]
                      [ 0.00117395 -0.00371071  1.09159451]]
Derived NPM -1     : [[ 1.06349549  0.00640891 -0.01580679]
                      [-0.49207413  1.36822341  0.09133709]
                      [-0.00281646  0.00464417  0.91641857]]
Use Derived NPM    : True
Use Derived NPM -1 : True
```

copy()

Returns a copy of the *RGB* colourspace.

Returns *RGB* colourspace copy.

Return type *RGB_Colourspace*

property name

Getter and setter property for the name.

Parameters **value** (`unicode`) – Value to set the name with.

Returns *RGB* colourspace name.

Return type unicode

property primaries
Getter and setter property for the primaries.

Parameters **value** (array_like) – Value to set the primaries with.

Returns *RGB* colourspace primaries.

Return type array_like

property use_derived_RGB_to_XYZ_matrix
Getter and setter property for whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Parameters **value** (bool) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Returns Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.

Return type bool

property use_derived_XYZ_to_RGB_matrix
Getter and setter property for Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Parameters **value** (bool) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Returns Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

Return type bool

use_derived_transformation_matrices(*usage=True*)
Enables or disables usage of both derived transformations matrices, the normalised primary matrix and its inverse in subsequent computations.

Parameters **usage** (bool, optional) – Whether to use the derived transformations matrices.

Returns Definition success.

Return type bool

property whitepoint
Getter and setter property for the whitepoint.

Parameters **value** (array_like) – Value to set the whitepoint with.

Returns *RGB* colourspace whitepoint.

Return type array_like

property whitepoint_name
Getter and setter property for the whitepoint_name.

Parameters **value** (unicode) – Value to set the whitepoint_name with.

Returns *RGB* colourspace whitepoint name.

Return type unicode

RGB_COLOURSPACES	Aggregated *RGB* colourspaces.

colour.RGB_COLOURSPACES

colour.**RGB_COLOURSPACES = CaseInsensitiveMapping({'ACES2065-1': ..., 'ACEScc': ..., 'ACEScct': ..., 'AC**

Aggregated *RGB* colourspaces.

RGB_COLOURSPACES : CaseInsensitiveMapping

Aliases:

- 'aces': ACES_2065_1_COLOURSPACE.name
- 'adobe1998': ADOBE_RGB_1998_COLOURSPACE.name
- 'prophoto': PROPHOTO_RGB_COLOURSPACE.name

colour.models

ACES_2065_1_COLOURSPACE	*ACES2065-1* colourspace, base encoding, used for exchange of full fidelity images and archiving.
ACES_CC_COLOURSPACE	*ACEScc* colourspace, a working space for color correctors, target for ASC-CDL values created onset.
ACES_CCT_COLOURSPACE	*ACEScct* colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.
ACES_PROXY_COLOURSPACE	*ACESproxy* colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management.
ACES_CG_COLOURSPACE	*ACEScg* colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.
ADOBE_RGB_1998_COLOURSPACE	*Adobe RGB (1998)* colourspace.
ADOBE_WIDE_GAMUT_RGB_COLOURSPACE	*Adobe Wide Gamut RGB* colourspace.
ALEXA_WIDE_GAMUT_COLOURSPACE	*ARRI ALEXA Wide Gamut* colourspace.
APPLE_RGB_COLOURSPACE	*Apple RGB* colourspace.
BEST_RGB_COLOURSPACE	*Best RGB* colourspace.
BETA_RGB_COLOURSPACE	*Beta RGB* colourspace.
BT470_525_COLOURSPACE	*ITU-R BT.470 - 525* colourspace.
BT470_625_COLOURSPACE	*ITU-R BT.470 - 625* colourspace.
BT709_COLOURSPACE	*ITU-R BT.709* colourspace.
BT2020_COLOURSPACE	*ITU-R BT.2020* colourspace.
CIE_RGB_COLOURSPACE	*CIE RGB* colourspace.
CINEMA_GAMUT_COLOURSPACE	*Canon Cinema Gamut* colourspace.
COLOR_MATCH_RGB_COLOURSPACE	*ColorMatch RGB* colourspace.
DCDM_XYZ_COLOURSPACE	*DCDM XYZ* colourspace.
DCI_P3_COLOURSPACE	*DCI-P3* colourspace.
DCI_P3_P_COLOURSPACE	*DCI-P3+* colourspace.
DISPLAY_P3_COLOURSPACE	*Display P3* colourspace.
DON_RGB_4_COLOURSPACE	*Don RGB 4* colourspace.
ECI_RGB_V2_COLOURSPACE	*ECI RGB v2* colourspace.
EKTA_SPACE_PS_5_COLOURSPACE	*Ekta Space PS 5* colourspace.
F_GAMUT_COLOURSPACE	*Fujifilm F-Gamut* colourspace.
PROTUNE_NATIVE_COLOURSPACE	*Protune Native* colourspace.
MAX_RGB_COLOURSPACE	*Max RGB* colourspace.
NTSC_1953_COLOURSPACE	*NTSC (1953)* colourspace.

Continued on next page

Table 168 – continued from previous page

NTSC_1987_COLOURSPACE	*NTSC (1987)* colourspace.
P3_D65_COLOURSPACE	*P3-D65* colourspace.
PAL_SECAM_COLOURSPACE	*Pal/Secam* colourspace.
RED_COLOR_COLOURSPACE	*REDcolor* colourspace.
RED_COLOR_2_COLOURSPACE	*REDcolor2* colourspace.
RED_COLOR_3_COLOURSPACE	*REDcolor3* colourspace.
RED_COLOR_4_COLOURSPACE	*REDcolor4* colourspace.
RED_WIDE_GAMUT_RGB_COLOURSPACE	*REDWideGamutRGB* colourspace.
DRAGON_COLOR_COLOURSPACE	*DRAGONcolor* colourspace.
DRAGON_COLOR_2_COLOURSPACE	*DRAGONcolor2* colourspace.
ROMM_RGB_COLOURSPACE	*ROMM RGB* colourspace.
RIMM_RGB_COLOURSPACE	*RIMM RGB* colourspace.
ERIMM_RGB_COLOURSPACE	*ERIMM RGB* colourspace.
PROPHOTO_RGB_COLOURSPACE	*ProPhoto RGB* colourspace, an alias colourspace for *ROMM RGB*.
RUSSELL_RGB_COLOURSPACE	*Russell RGB* colourspace.
SMPTE_240M_COLOURSPACE	*SMPTE 240M* colourspace.
SMPTE_C_COLOURSPACE	Implements support for the *RGB* colourspaces datasets from `colour.models.datasets.aces_rgb`, etc....
S_GAMUT_COLOURSPACE	*S-Gamut* colourspace.
S_GAMUT3_COLOURSPACE	*S-Gamut3* colourspace.
S_GAMUT3_CINE_COLOURSPACE	*S-Gamut3.Cine* colourspace.
sRGB_COLOURSPACE	*sRGB* colourspace.
V_GAMUT_COLOURSPACE	*Panasonic V-Gamut* colourspace.
XTREME_RGB_COLOURSPACE	*Xtreme RGB* colourspace.

colour.models.ACES_2065_1_COLOURSPACE

colour.models.**ACES_2065_1_COLOURSPACE = RGB_Colourspace(ACES2065-1, [[7.34700000e-01, 2.65300000e-01], [**

ACES2065-1 colourspace, base encoding, used for exchange of full fidelity images and archiving.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

ACES_2065_1_COLOURSPACE : RGB_Colourspace

colour.models.ACES_CC_COLOURSPACE

colour.models.**ACES_CC_COLOURSPACE = RGB_Colourspace(ACEScc, [[0.713, 0.293], [0.165, 0.83], [0.128, 0**

ACEScc colourspace, a working space for color correctors, target for ASC-CDL values created on-set.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

ACES_CC_COLOURSPACE : RGB_Colourspace

colour.models.ACES_CCT_COLOURSPACE

colour.models.**ACES_CCT_COLOURSPACE = RGB_Colourspace(ACEScct, [[0.713, 0.293], [0.165, 0.83], [0.128,**

ACEScct colourspace, an alternative working space for colour correctors, intended to be transient and internal to software or hardware systems, and is specifically not intended for interchange or archiving.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

ACES_CCT_COLOURSPACE : RGB_Colourspace

colour.models.ACES_PROXY_COLOURSPACE

colour.models.**ACES_PROXY_COLOURSPACE = RGB_Colourspace(ACESproxy, [[0.713, 0.293], [0.165, 0.83], [0.**

ACESproxy colourspace, a lightweight encoding for transmission over HD-SDI (or other production transmission schemes), onset look management. Not intended to be stored or used in production imagery or for final colour grading / mastering.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

ACES_PROXY_COLOURSPACE : RGB_Colourspace

colour.models.ACES_CG_COLOURSPACE

colour.models.**ACES_CG_COLOURSPACE = RGB_Colourspace(ACEScg, [[0.713, 0.293], [0.165, 0.83], [0.128, 0**
ACEScg colourspace, a working space for paint/compositor applications that don't support ACES2065-1 or ACEScc.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

ACES_CG_COLOURSPACE : RGB_Colourspace

colour.models.ADOBE_RGB_1998_COLOURSPACE

colour.models.**ADOBE_RGB_1998_COLOURSPACE = RGB_Colourspace(Adobe RGB (1998), [[0.64, 0.33], [0.21, 0.71**
Adobe RGB (1998) colourspace.

References

[AdobeSystems05]

ADOBE_RGB_1998_COLOURSPACE : RGB_Colourspace

colour.models.ADOBE_WIDE_GAMUT_RGB_COLOURSPACE

colour.models.**ADOBE_WIDE_GAMUT_RGB_COLOURSPACE = RGB_Colourspace(Adobe Wide Gamut RGB, [[0.7347, 0.2653]**
Adobe Wide Gamut RGB colourspace.

References

[Wik04d]

ADOBE_WIDE_GAMUT_RGB_COLOURSPACE : RGB_Colourspace

colour.models.ALEXA_WIDE_GAMUT_COLOURSPACE

colour.models.**ALEXA_WIDE_GAMUT_COLOURSPACE = RGB_Colourspace(ALEXA Wide Gamut, [[0.684 , 0.313], [0.22**
ARRI ALEXA Wide Gamut colourspace.

References

[ARR12]

ALEXA_WIDE_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.APPLE_RGB_COLOURSPACE

colour.models.**APPLE_RGB_COLOURSPACE = RGB_Colourspace(Apple RGB, [[0.625, 0.34], [0.28 , 0.595], [0.1**
Apple RGB colourspace.

References

[SBS99]

APPLE_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BEST_RGB_COLOURSPACE

colour.models.**BEST_RGB_COLOURSPACE = RGB_Colourspace(Best RGB, [[0.73519164, 0.26480836], [0.21533613,**
Best RGB colourspace.

References

[Huta]

BEST_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BETA_RGB_COLOURSPACE

colour.models.**BETA_RGB_COLOURSPACE = RGB_Colourspace(Beta RGB, [[0.6888, 0.3112], [0.1986, 0.7551], [0**
Beta RGB colourspace.

References

[Lin14]

BETA_RGB_COLOURSPACE : RGB_Colourspace

colour.models.BT470_525_COLOURSPACE

colour.models.**BT470_525_COLOURSPACE = RGB_Colourspace(ITU-R BT.470 - 525, [[0.67, 0.33], [0.21, 0.71],**
ITU-R BT.470 - 525 colourspace.

References

[InternationalTUnion98]

BT470_525_COLOURSPACE : RGB_Colourspace

colour.models.BT470_625_COLOURSPACE

colour.models.**BT470_625_COLOURSPACE = RGB_Colourspace(ITU-R BT.470 - 625, [[0.64, 0.33], [0.29, 0.6],**
ITU-R BT.470 - 625 colourspace.

References

[InternationalTUnion98]

BT470_625_COLOURSPACE : RGB_Colourspace

colour.models.BT709_COLOURSPACE

colour.models.**BT709_COLOURSPACE = RGB_Colourspace(ITU-R BT.709, [[0.64, 0.33], [0.3 , 0.6], [0.15, 0.**
ITU-R BT.709 colourspace.

References

[InternationalTUnion15b]

BT709_COLOURSPACE : RGB_Colourspace

colour.models.BT2020_COLOURSPACE

colour.models.**BT2020_COLOURSPACE = RGB_Colourspace(ITU-R BT.2020, [[0.708, 0.292], [0.17 , 0.797], [0.**
ITU-R BT.2020 colourspace.

References

[InternationalTUnion15a]

BT2020_COLOURSPACE : RGB_Colourspace

colour.models.CIE_RGB_COLOURSPACE

colour.models.**CIE_RGB_COLOURSPACE = RGB_Colourspace(CIE RGB, [[0.73474284, 0.26525716], [0.27377903, 0.**
CIE RGB colourspace.

References

[FBH97]

CIE_RGB_COLOURSPACE : RGB_Colourspace

colour.models.CINEMA_GAMUT_COLOURSPACE

colour.models.**CINEMA_GAMUT_COLOURSPACE = RGB_Colourspace(Cinema Gamut, [[0.74, 0.27], [0.17, 1.14], [0**
Canon Cinema Gamut colourspace.

References

[Can14]

CINEMA_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.COLOR_MATCH_RGB_COLOURSPACE

colour.models.**COLOR_MATCH_RGB_COLOURSPACE = RGB_Colourspace(ColorMatch RGB, [[0.63 , 0.34], [0.295, 0.**
ColorMatch RGB colourspace.

References

[Lin14]

COLOR_MATCH_RGB_COLOURSPACE : RGB_Colourspace

colour.models.DCDM_XYZ_COLOURSPACE

colour.models.**DCDM_XYZ_COLOURSPACE = RGB_Colourspace(DCDM XYZ, [[1., 0.], [0., 1.], [0., 0.]], [0.333**
DCDM XYZ colourspace.

References

[DigitalCInitiatives07]

DCDM_XYZ_COLOURSPACE : RGB_Colourspace

colour.models.DCI_P3_COLOURSPACE

colour.models.**DCI_P3_COLOURSPACE = RGB_Colourspace(DCI-P3, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.**
DCI-P3 colourspace.

References

[DigitalCInitiatives07], [HewlettPDCompany09]

DCI_P3_COLOURSPACE : RGB_Colourspace

colour.models.DCI_P3_P_COLOURSPACE

colour.models.**DCI_P3_P_COLOURSPACE = RGB_Colourspace(DCI-P3+, [[0.74, 0.27], [0.22, 0.78], [0.09, -0.0**
DCI-P3+ colourspace.

References

[Can14]

DCI_P3_P_COLOURSPACE : RGB_Colourspace

colour.models.DISPLAY_P3_COLOURSPACE

colour.models.**DISPLAY_P3_COLOURSPACE = RGB_Colourspace(Display P3, [[0.68 , 0.32], [0.265, 0.69], [0**
Display P3 colourspace.

References

[AppleInc19]

DISPLAY_P3_COLOURSPACE : RGB_Colourspace

colour.models.DON_RGB_4_COLOURSPACE

colour.models.**DON_RGB_4_COLOURSPACE = RGB_Colourspace(Don RGB 4, [[0.69612069, 0.29956897], [0.21468298**
Don RGB 4 colourspace.

References

[Hutb]

DON_RGB_4_COLOURSPACE : RGB_Colourspace

colour.models.ECI_RGB_V2_COLOURSPACE

colour.models.**ECI_RGB_V2_COLOURSPACE = RGB_Colourspace(ECI RGB v2, [[0.67010309, 0.32989691], [0.209905**
ECI RGB v2 colourspace.

References

[EuropeanCInitiative02]

ECI_RGB_V2_COLOURSPACE : RGB_Colourspace

colour.models.EKTA_SPACE_PS_5_COLOURSPACE

colour.models.**EKTA_SPACE_PS_5_COLOURSPACE = RGB_Colourspace(Ekta Space PS 5, [[0.69473684, 0.30526316],**
Ekta Space PS 5 colourspace.

References

[Hol]

EKTA_SPACE_PS_5_COLOURSPACE : RGB_Colourspace

colour.models.F_GAMUT_COLOURSPACE

colour.models.**F_GAMUT_COLOURSPACE = RGB_Colourspace(F-Gamut, [[0.708, 0.292], [0.17 , 0.797], [0.131,**
Fujifilm F-Gamut colourspace.

References

[Fuj16]

F_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.PROTUNE_NATIVE_COLOURSPACE

colour.models.**PROTUNE_NATIVE_COLOURSPACE = RGB_Colourspace(Protune Native, [[0.69848046, 0.19302645], [**
Protune Native colourspace.

References

[GDM16], [Man15]

PROTUNE_NATIVE_COLOURSPACE : RGB_Colourspace

colour.models.MAX_RGB_COLOURSPACE

colour.models.**MAX_RGB_COLOURSPACE = RGB_Colourspace(Max RGB, [[0.73413379, 0.26586621], [0.10039113, 0.**
Max RGB colourspace.

References

[Hutc]

MAX_RGB_COLOURSPACE : RGB_Colourspace

colour.models.NTSC_1953_COLOURSPACE

colour.models.**NTSC_1953_COLOURSPACE = RGB_Colourspace(NTSC (1953), [[0.67, 0.33], [0.21, 0.71], [0.14,**
NTSC (1953) colourspace.

References

[InternationalTUnion98]

NTSC_1953_COLOURSPACE : RGB_Colourspace

colour.models.NTSC_1987_COLOURSPACE

colour.models.**NTSC_1987_COLOURSPACE = RGB_Colourspace(NTSC (1987), [[0.63 , 0.34], [0.31 , 0.595], [0**
NTSC (1987) colourspace.

References

[SocietyoMPaTEngineers04]

NTSC_1987_COLOURSPACE : RGB_Colourspace

colour.models.P3_D65_COLOURSPACE

colour.models.**P3_D65_COLOURSPACE = RGB_Colourspace(P3-D65, [[0.68 , 0.32], [0.265, 0.69], [0.15 , 0.**
P3-D65 colourspace.

P3_D65_COLOURSPACE : RGB_Colourspace

colour.models.PAL_SECAM_COLOURSPACE

colour.models.**PAL_SECAM_COLOURSPACE = RGB_Colourspace(Pal/Secam, [[0.64, 0.33], [0.29, 0.6], [0.15, 0**
Pal/Secam colourspace.

References

[InternationalTUnion98]

PAL_SECAM_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_COLOURSPACE

colour.models.**RED_COLOR_COLOURSPACE = RGB_Colourspace(REDcolor, [[0.70105856, 0.33018098], [0.29881132,**
REDcolor colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_2_COLOURSPACE

colour.models.**RED_COLOR_2_COLOURSPACE = RGB_Colourspace(REDcolor2, [[0.89740722, 0.33077623], [0.296022**
REDcolor2 colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_2_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_3_COLOURSPACE

colour.models.**RED_COLOR_3_COLOURSPACE = RGB_Colourspace(REDcolor3, [[0.70259866, 0.33018559], [0.295782**
REDcolor3 colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_3_COLOURSPACE : RGB_Colourspace

colour.models.RED_COLOR_4_COLOURSPACE

colour.models.**RED_COLOR_4_COLOURSPACE = RGB_Colourspace(REDcolor4, [[0.70259815, 0.3301851], [0.295782**
REDcolor4 colourspace.

References

[Man15], [SonyImageworks12]

RED_COLOR_4_COLOURSPACE : RGB_Colourspace

colour.models.RED_WIDE_GAMUT_RGB_COLOURSPACE

colour.models.**RED_WIDE_GAMUT_RGB_COLOURSPACE = RGB_Colourspace(REDWideGamutRGB, [[0.780308, 0.304253], [**
REDWideGamutRGB colourspace.

References

[Man15], [Nat16], [SonyImageworks12]

RED_WIDE_GAMUT_RGB_COLOURSPACE : RGB_Colourspace

colour.models.DRAGON_COLOR_COLOURSPACE

colour.models.**DRAGON_COLOR_COLOURSPACE = RGB_Colourspace(DRAGONcolor, [[0.75865589, 0.33035535], [0.294**
DRAGONcolor colourspace.

References

[Man15], [SonyImageworks12]

DRAGON_COLOR_COLOURSPACE : RGB_Colourspace

colour.models.DRAGON_COLOR_2_COLOURSPACE

colour.models.**DRAGON_COLOR_2_COLOURSPACE = RGB_Colourspace(DRAGONcolor2, [[0.75865621, 0.33035584], [0.**
DRAGONcolor2 colourspace.

References

[Man15], [SonyImageworks12]

DRAGON_COLOR_2_COLOURSPACE : RGB_Colourspace

colour.models.ROMM_RGB_COLOURSPACE

colour.models.**ROMM_RGB_COLOURSPACE = RGB_Colourspace(ROMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.59**
ROMM RGB colourspace.

References

[ANS03], [SWG00]

ROMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.RIMM_RGB_COLOURSPACE

colour.models.**RIMM_RGB_COLOURSPACE = RGB_Colourspace(RIMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.59**
RIMM RGB colourspace. In cases in which it is necessary to identify a specific precision level, the notation *RIMM8 RGB*, *RIMM12 RGB* and *RIMM16 RGB* is used.

References

[SWG00]

RIMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.ERIMM_RGB_COLOURSPACE

colour.models.**ERIMM_RGB_COLOURSPACE = RGB_Colourspace(ERIMM RGB, [[7.34700000e-01, 2.65300000e-01], [1.**
ERIMM RGB colourspace.

References

[SWG00]

ERIMM_RGB_COLOURSPACE : RGB_Colourspace

colour.models.PROPHOTO_RGB_COLOURSPACE

colour.models.**PROPHOTO_RGB_COLOURSPACE = RGB_Colourspace(ProPhoto RGB, [[7.34700000e-01, 2.65300000e-01]**
ProPhoto RGB colourspace, an alias colourspace for *ROMM RGB*.

References

[ANS03], [SWG00]

PROPHOTO_RGB_COLOURSPACE : RGB_Colourspace

colour.models.RUSSELL_RGB_COLOURSPACE

colour.models.**RUSSELL_RGB_COLOURSPACE = RGB_Colourspace(Russell RGB, [[0.69, 0.31], [0.18, 0.77], [0.1**
Russell RGB colourspace.

References

[Cot]

RUSSELL_RGB_COLOURSPACE : RGB_Colourspace

colour.models.SMPTE_240M_COLOURSPACE

colour.models.**SMPTE_240M_COLOURSPACE = RGB_Colourspace(SMPTE 240M, [[0.63 , 0.34], [0.31 , 0.595], [0**
SMPTE 240M colourspace.

References

[SocietyoMPaTEngineers99],

SMPTE_240M_COLOURSPACE : RGB_Colourspace

colour.models.SMPTE_C_COLOURSPACE

colour.models.**SMPTE_C_COLOURSPACE = RGB_Colourspace(SMPTE C, [[0.63 , 0.34], [0.31 , 0.595], [0.155,**

Implements support for the *RGB* colourspaces datasets from `colour.models.datasets.aces_rgb`, etc. . . .

Colour science literature related to *RGB* colourspaces and encodings defines their dataset using different degree of precision or rounding. While instances where a whitepoint is being defined with a value different than its canonical agreed one are rare, it is however very common to have normalised primary matrices rounded at different decimals. This can yield large discrepancies in computations.

Such an occurrence is the *V-Gamut* colourspace white paper, that defines the *V-Gamut* to *ITU-R BT.709* conversion matrix as follows:

```
[[ 1.806576 -0.695697 -0.110879]
 [-0.170090  1.305955 -0.135865]
 [-0.025206 -0.154468  1.179674]]
```

Computing this matrix using *ITU-R BT.709* colourspace derived normalised primary matrix yields:

```
[[ 1.8065736 -0.6956981 -0.1108786]
 [-0.1700890  1.3059548 -0.1358648]
 [-0.0252057 -0.1544678  1.1796737]]
```

The latter matrix is almost equals with the former, however performing the same computation using *IEC 61966-2-1:1999 sRGB* colourspace normalised primary matrix introduces severe disparities:

```
[[ 1.8063853 -0.6956147 -0.1109453]
 [-0.1699311  1.3058387 -0.1358616]
 [-0.0251630 -0.1544899  1.1797117]]
```

In order to provide support for both literature defined dataset and accurate computations enabling transformations without loss of precision, the `colour.RGB_Colourspace` class provides two sets of transformation matrices:

- Instantiation transformation matrices
- Derived transformation matrices

Upon instantiation, the `colour.RGB_Colourspace` class stores the given `RGB_to_XYZ_matrix` and `XYZ_to_RGB_matrix` arguments and also computes their derived counterpart using the `primaries` and `whitepoint` arguments.

Whether the initialisation or derived matrices are used in subsequent computations is dependent on the `colour.RGB_Colourspace.use_derived_RGB_to_XYZ_matrix` and `colour.RGB_Colourspace.use_derived_XYZ_to_RGB_matrix` attributes values.

Parameters

- **name** (`unicode`) – *RGB* colourspace name.
- **primaries** (`array_like`) – *RGB* colourspace primaries.
- **whitepoint** (`array_like`) – *RGB* colourspace whitepoint.
- **whitepoint_name** (`unicode, optional`) – *RGB* colourspace whitepoint name.

- **RGB_to_XYZ_matrix** (array_like, optional) – Transformation matrix from colourspace to *CIE XYZ* tristimulus values.
- **XYZ_to_RGB_matrix** (array_like, optional) – Transformation matrix from *CIE XYZ* tristimulus values to colourspace.
- **cctf_encoding** (object, optional) – Encoding colour component transfer function (Encoding CCTF) / opto-electronic transfer function (OETF / OECF) that maps estimated tristimulus values in a scene to $R'G'B'$ video component signal value.
- **cctf_decoding** (object, optional) – Decoding colour component transfer function (Decoding CCTF) / electro-optical transfer function (EOTF / EOCF) that maps an $R'G'B'$ video component signal value to tristimulus values at the display.
- **use_derived_RGB_to_XYZ_matrix** (bool, optional) – Whether to use the instantiation time normalised primary matrix or to use a computed derived normalised primary matrix.
- **use_derived_XYZ_to_RGB_matrix** (bool, optional) – Whether to use the instantiation time inverse normalised primary matrix or to use a computed derived inverse normalised primary matrix.

colour.models.**name**

colour.models.**primaries**

colour.models.**whitepoint**

colour.models.**whitepoint_name**

colour.models.**RGB_to_XYZ_matrix**

colour.models.**XYZ_to_RGB_matrix**

colour.models.**cctf_encoding**

colour.models.**cctf_decoding**

colour.models.**use_derived_RGB_to_XYZ_matrix**

colour.models.**use_derived_XYZ_to_RGB_matrix**

colour.models.**__str__**()

colour.models.**__repr__**()

colour.models.**use_derived_transformation_matrices**()

colour.models.**chromatically_adapt**()

colour.models.**copy**()

Notes

- The normalised primary matrix defined by colour.RGB_Colourspace.RGB_to_XYZ_matrix attribute is treated as the prime matrix from which the inverse will be calculated as required by the internal derivation mechanism. This behaviour has been chosen in accordance with literature where commonly a *RGB* colourspace is defined by its normalised primary matrix as it is directly computed from the chosen primaries and whitepoint.

References

[InternationalECommission99], [Pan14]

Examples

```
>>> p = np.array([0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700])
>>> whitepoint = np.array([0.32168, 0.33767])
>>> RGB_to_XYZ_matrix = np.identity(3)
>>> XYZ_to_RGB_matrix = np.identity(3)
>>> colourspace = RGB_Colourspace('RGB Colourspace', p, whitepoint, 'ACES',
...                               RGB_to_XYZ_matrix, XYZ_to_RGB_matrix)
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_transformation_matrices(True)
True
>>> colourspace.RGB_to_XYZ_matrix
array([[  9.5255239...e-01,   0.0000000...e+00,   9.3678631...e-05],
       [  3.4396645...e-01,   7.2816609...e-01,  -7.2132546...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   1.0088251...e+00]])
>>> colourspace.XYZ_to_RGB_matrix
array([[  1.0498110...e+00,   0.0000000...e+00,  -9.7484540...e-05],
       [ -4.9590302...e-01,   1.3733130...e+00,   9.8240036...e-02],
       [  0.0000000...e+00,   0.0000000...e+00,   9.9125201...e-01]])
>>> colourspace.use_derived_RGB_to_XYZ_matrix = False
>>> colourspace.RGB_to_XYZ_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> colourspace.use_derived_XYZ_to_RGB_matrix = False
>>> colourspace.XYZ_to_RGB_matrix
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

colour.models.S_GAMUT_COLOURSPACE

colour.models.**S_GAMUT_COLOURSPACE = RGB_Colourspace(S-Gamut, [[0.73 , 0.28], [0.14 , 0.855], [0.1 , -**

S-Gamut colourspace.

References

[GDY+], [SonyCorporationb]

S_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.S_GAMUT3_COLOURSPACE

colour.models.**S_GAMUT3_COLOURSPACE = RGB_Colourspace(S-Gamut3, [[0.73 , 0.28], [0.14 , 0.855], [0.1 ,**
S-Gamut3 colourspace.

References

[SonyCorporationc]

S_GAMUT3_COLOURSPACE : RGB_Colourspace

colour.models.S_GAMUT3_CINE_COLOURSPACE

colour.models.**S_GAMUT3_CINE_COLOURSPACE = RGB_Colourspace(S-Gamut3.Cine, [[0.766, 0.275], [0.225, 0.8]**
S-Gamut3.Cine colourspace.

References

[SonyCorporationa]

S_GAMUT3_CINE_COLOURSPACE : RGB_Colourspace

colour.models.sRGB_COLOURSPACE

colour.models.**sRGB_COLOURSPACE = RGB_Colourspace(sRGB, [[0.64, 0.33], [0.3 , 0.6], [0.15, 0.06]], [0**
sRGB colourspace.

References

[InternationalECommission99], [InternationalTUnion15b]

sRGB_COLOURSPACE : RGB_Colourspace

colour.models.V_GAMUT_COLOURSPACE

colour.models.**V_GAMUT_COLOURSPACE = RGB_Colourspace(V-Gamut, [[0.73 , 0.28], [0.165, 0.84], [0.1 , -**
Panasonic V-Gamut colourspace.

References

[Pan14]

V_GAMUT_COLOURSPACE : RGB_Colourspace

colour.models.XTREME_RGB_COLOURSPACE

colour.models.**XTREME_RGB_COLOURSPACE = RGB_Colourspace(Xtreme RGB, [[1., 0.], [0., 1.], [0., 0.]], [0**

Xtreme RGB colourspace.

References

[Hutd]

XTREME_RGB_COLOURSPACE : RGB_Colourspace

Colour Component Transfer Functions

colour

cctf_encoding(value[, function])	Encodes linear RGB values to non linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).
CCTF_ENCODINGS	Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES attributes, the colour.models.cctf_encoding_ProPhotoRGB(), colour.models.cctf_encoding_RIMMRGB(), colour.models.cctf_encoding_ROMMRGB() definitions and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).
cctf_decoding(value[, function])	Decodes non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).
CCTF_DECODINGS	Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by colour.LOG_DECODINGS, colour.EOTFS, colour.OETF_INVERSES attributes, the colour.models.cctf_decoding_ProPhotoRGB(), colour.models.cctf_decoding_RIMMRGB(), colour.models.cctf_decoding_ROMMRGB() definitions and 3 gamma decoding functions (2.2, 2.4, 2.6).

colour.cctf_encoding

colour.**cctf_encoding**(*value*, *function='sRGB'*, ***kwargs*)

Encodes linear RGB values to non linear $R'G'B'$ values using given encoding colour component transfer function (Encoding CCTF).

Parameters

- **value** (numeric or array_like) – Linear RGB values.
- **function** (unicode, optional) – {colour.CCTF_ENCODINGS}, Computation function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for the relevant encoding CCTF of the colour.CCTF_ENCODINGS attribute collection.

Warning: For *ITU-R BT.2100*, only the inverse electro-optical transfer functions (EOTFs / EOCFs) are exposed by this definition, please refer to the colour.oetf() definition for the opto-electronic transfer functions (OETF / OECF).

Returns Non linear $R'G'B'$ values.

Return type numeric or ndarray

Examples

```
>>> cctf_encoding(0.18, function='PLog', log_reference=400)
...
0.3910068...
>>> cctf_encoding(0.18, function='ST 2084', L_p=1000)
...
0.1820115...
>>> cctf_encoding(
...     0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.CCTF_ENCODINGS

colour.**CCTF_ENCODINGS = CaseInsensitiveMapping({'Gamma 2.2': ..., 'Gamma 2.4': ..., 'Gamma 2.6': ..., '**

Supported encoding colour component transfer functions (Encoding CCTFs), a collection of the functions defined by colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES attributes, the colour.models.cctf_encoding_ProPhotoRGB(), colour.models.cctf_encoding_RIMMRGB(), colour.models.cctf_encoding_ROMMRGB() definitions and 3 gamma encoding functions (1 / 2.2, 1 / 2.4, 1 / 2.6).

Warning: For *ITU-R BT.2100*, only the inverse electro-optical transfer functions (EOTFs / EOCFs) are exposed by this attribute, please refer to the colour.OETFS attribute for the opto-electronic transfer functions (OETF / OECF).

CCTF_ENCODINGS [CaseInsensitiveMapping] {colour.LOG_ENCODINGS, colour.OETFS, colour.EOTF_INVERSES}

colour.cctf_decoding

colour.**cctf_decoding**(*value*, *function='sRGB'*, ***kwargs*)

Decodes non-linear $R'G'B'$ values to linear RGB values using given decoding colour component transfer function (Decoding CCTF).

Parameters

- **value** (numeric or array_like) – Non-linear $R'G'B'$ values.
- **function** (unicode, optional) – {colour.CCTF_DECODINGS}, Computation function.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for the relevant decoding CCTF of the colour.CCTF_DECODINGS attribute collection.

Warning: For *ITU-R BT.2100*, only the electro-optical transfer functions (EOTFs / EOCFs) are exposed by this definition, please refer to the colour.oetf_inverse() definition for the inverse opto-electronic transfer functions (OETF / OECF).

Returns Linear RGB values.

Return type numeric or ndarray

Examples

```
>>> cctf_decoding(0.391006842619746, function='PLog', log_reference=400)
...
0.1...
>>> cctf_decoding(0.182011532850008, function='ST 2084', L_p=1000)
...
0.1...
>>> cctf_decoding(
...     0.461356129500442, function='ITU-R BT.1886')
0.1...
```

colour.CCTF_DECODINGS

colour.**CCTF_DECODINGS = CaseInsensitiveMapping({'Gamma 2.2': ..., 'Gamma 2.4': ..., 'Gamma 2.6': ..., '**

Supported decoding colour component transfer functions (Decoding CCTFs), a collection of the functions defined by colour.LOG_DECODINGS, colour.EOTFS, colour.OETF_INVERSES attributes, the colour.models.cctf_decoding_ProPhotoRGB(), colour.models.cctf_decoding_RIMMRGB(), colour.models.cctf_decoding_ROMMRGB() definitions and 3 gamma decoding functions (2.2, 2.4, 2.6).

Warning: For *ITU-R BT.2100*, only the electro-optical transfer functions (EOTFs / EOCFs) are exposed by this attribute, please refer to the colour.OETF_INVERSES attribute for the inverse opto-electronic transfer functions (OETF / OECF).

Notes

- The order by which this attribute is defined and updated is critically important to ensure that *ITU-R BT.2100* definitions are reciprocal.

CCTF_DECODINGS [CaseInsensitiveMapping] {colour.LOG_DECODINGS, colour.EOTFS, colour.OETF_INVERSES}

colour.models

cctf_encoding_ROMMRGB(X[, bit_depth, out_int])	Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_ROMMRGB(X_p[, bit_depth, in_int])	Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).
cctf_encoding_RIMMRGB(X[, bit_depth, ...])	Defines the *RIMM RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_RIMMRGB(X_p[, bit_depth, ...])	Defines the *RIMM RGB* decoding colour component transfer function (Encoding CCTF).

colour.models.cctf_encoding_ROMMRGB

colour.models.**cctf_encoding_ROMMRGB**(*X*, *bit_depth=8*, *out_int=False*)

Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ROMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Non-linear data X'_{ROMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> cctf_encoding_ROMMRGB(0.18)
0.3857114...
>>> cctf_encoding_ROMMRGB(0.18, out_int=True)
98
```

colour.models.cctf_decoding_ROMMRGB

colour.models.**cctf_decoding_ROMMRGB**(*X_p*, *bit_depth=8*, *in_int=False*)

Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ROMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns Linear data X_{ROMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> cctf_decoding_ROMMRGB(0.385711424751138)
0.1...
>>> cctf_decoding_ROMMRGB(98, in_int=True)
0.1...
```

colour.models.cctf_encoding_RIMMRGB

colour.models.**cctf_encoding_RIMMRGB**(*X*, *bit_depth=8*, *out_int=False*, *E_clip=2.0*)

Defines the *RIMM RGB* encoding colour component transfer function (Encoding CCTF).

RIMM RGB encoding non-linearity is based on that specified by *Recommendation ITU-R BT.709-6*.

Parameters

- **X** (numeric or array_like) – Linear data X_{RIMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.
- **E_clip** (numeric, optional) – Maximum exposure level.

Returns Non-linear data X'_{RIMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> cctf_encoding_RIMMRGB(0.18)
0.2916737...
>>> cctf_encoding_RIMMRGB(0.18, out_int=True)
74
```

colour.models.cctf_decoding_RIMMRGB

colour.models.**cctf_decoding_RIMMRGB**(*X_p*, *bit_depth=8*, *in_int=False*, *E_clip=2.0*)

Defines the *RIMM RGB* decoding colour component transfer function (Encoding CCTF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{RIMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **E_clip** (numeric, optional) – Maximum exposure level.

Returns Linear data X_{RIMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> cctf_decoding_RIMMRGB(0.291673732475746)
0.1...
>>> cctf_decoding_RIMMRGB(74, in_int=True)
0.1...
```

Aliases

colour.models

cctf_encoding_ProPhotoRGB(X[, bit_depth, ...])	Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).
cctf_decoding_ProPhotoRGB(X_p[, bit_depth, ...])	Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

colour.models.cctf_encoding_ProPhotoRGB

colour.models.**cctf_encoding_ProPhotoRGB**(*X*, *bit_depth=8*, *out_int=False*)

Defines the *ProPhoto RGB* encoding colour component transfer function (Encoding CCTF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ROMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Non-linear data X'_{ROMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> cctf_encoding_ROMMRGB(0.18)
0.3857114...
>>> cctf_encoding_ROMMRGB(0.18, out_int=True)
98
```

colour.models.cctf_decoding_ProPhotoRGB

colour.models.**cctf_decoding_ProPhotoRGB**(*X_p*, *bit_depth=8*, *in_int=False*)

Defines the *ProPhoto RGB* decoding colour component transfer function (Encoding CCTF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ROMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns Linear data X_{ROMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[ANS03], [SWG00]

Examples

```
>>> cctf_decoding_ROMMRGB(0.385711424751138)
0.1...
>>> cctf_decoding_ROMMRGB(98, in_int=True)
0.1...
```

Opto-Electronic Transfer Functions

colour

oetf(value[, function])	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF / OECF).
OETFS	Supported opto-electrical transfer functions (OETFs / OECFs).
oetf_inverse(value[, function])	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given inverse opto-electronic transfer function (OETF / OECF).
OETF_INVERSES	Supported inverse opto-electrical transfer functions (OETFs / OECFs).

colour.oetf

colour.**oetf**(*value*, *function='ITU-R BT.709'*, ***kwargs*)

Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.709', 'ARIB STD-B67', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'SMPTE 240M', 'ST 2084'}**, Opto-electronic transfer function (OETF / OECF).

Other Parameters

- **E_clip** (*numeric, optional*) – {colour.models.cctf_encoding_RIMMRGB()}, Maximum exposure level.
- **I_max** (*numeric, optional*) – {colour.models.cctf_encoding_ROMMRGB(), colour.models.cctf_encoding_RIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **is_12_bits_system** (*bool, optional*) – {colour.models.oetf_BT2020()}, *ITU-R BT.2020 alpha* and *beta* constants are used if system is not 12-bit.
- **r** (*numeric, optional*) – {colour.models.oetf_ARIBSTDB67()}, Video level corresponding to reference white level.

Returns $R'G'B'$ video component signal value.

Return type numeric or ndarray

Examples

```
>>> oetf(0.18)
0.4090077...
>>> oetf(0.18, function='ITU-R BT.601')
0.4090077...
```

colour.OETFS

colour.**OETFS = CaseInsensitiveMapping({'ARIB STD-B67': ..., 'ITU-R BT.2020': ..., 'ITU-R BT.2100 HLG': .**
Supported opto-electrical transfer functions (OETFs / OECFs).

OETFS [CaseInsensitiveMapping] **{'sRGB', 'ARIB STD-B67', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709', 'SMPTE 240M', 'ST 2084'}**

colour.oetf_inverse

colour.**oetf_inverse**(*value*, *function='ITU-R BT.709'*, ***kwargs*)
Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given inverse opto-electronic transfer function (OETF / OECF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.709', 'ARIB STD-B67', 'ITU-R BT.2100 HLD', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', }**, Inverse opto-electronic transfer function (OETF / OECF).

Other Parameters **r** (*numeric, optional*) – {colour.models.oetf_ARIBSTDB67()}, Video level corresponding to reference white level.

Returns Tristimulus values at the display.

Return type numeric or ndarray

Examples

```
>>> oetf_inverse(0.409007728864150)
0.1...
>>> oetf_inverse(
...     0.409007728864150, function='ITU-R BT.601')
0.1...
```

colour.OETF_INVERSES

colour.**OETF_INVERSES = CaseInsensitiveMapping({'ARIB STD-B67': ..., 'ITU-R BT.2100 HLD': ..., 'ITU-R BT.2**
Supported inverse opto-electrical transfer functions (OETFs / OECFs).

OETF_INVERSES [CaseInsensitiveMapping] **{'ARIB STD-B67', 'ITU-R BT.2100 HLD', 'ITU-R BT.2100 PQ', 'ITU-R BT.601', 'ITU-R BT.709'}**

colour.models

oetf_ARIBSTDB67(E[, r, constants])	Defines *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF / OECF).
oetf_inverse_ARIBSTDB67(E_p[, r, constants])	Defines *ARIB STD-B67 (Hybrid Log-Gamma)* inverse opto-electrical transfer function (OETF / OECF).
oetf_BT2020(E[, is_12_bits_system, constants])	Defines *Recommendation ITU-R BT.2020* opto-electrical transfer function (OETF / OECF).
oetf_HLG_BT2100(E[, constants])	Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF / OECF).

Continued on next page

Table 173 – continued from previous page

oetf_inverse_HLG_BT2100(E_p[, constants])	Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-electrical transfer function (OETF / OECF).
oetf_PQ_BT2100(E)	Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF / OECF).
oetf_inverse_PQ_BT2100(E_p)	Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-electrical transfer function (OETF / OECF).
oetf_BT601(L)	Defines *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF / OECF).
oetf_inverse_BT601(E)	Defines *Recommendation ITU-R BT.601-7* inverse opto-electronic transfer function (OETF / OECF).
oetf_BT709(L)	Defines *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF / OECF).
oetf_inverse_BT709(V)	Defines *Recommendation ITU-R BT.709-6* inverse opto-electronic transfer function (OETF / OECF).
oetf_SMPTE240M(L_c)	Defines *SMPTE 240M* opto-electrical transfer function (OETF / OECF).

colour.models.oetf_ARIBSTDB67

colour.models.**oetf_ARIBSTDB67**(*E*, *r=0.5*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.55991073}*)

Defines *ARIB STD-B67 (Hybrid Log-Gamma)* opto-electrical transfer function (OETF / OECF).

Parameters

- **E** (numeric or array_like) – Voltage normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera color channel R, G, B.
- **r** (numeric, optional) – Video level corresponding to reference white level.
- **constants** (Structure, optional) – *ARIB STD-B67 (Hybrid Log-Gamma)* constants.

Returns Resulting non-linear signal E'.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

- This definition uses the *mirror* negative number handling mode of colour.models.gamma_function() definition to the sign of negative numbers.

References

[AssociationoRIaBusinesses15]

Examples

```
>>> oetf_ARIBSTDB67(0.18)
0.2121320...
```

colour.models.oetf_inverse_ARIBSTDB67

colour.models.**oetf_inverse_ARIBSTDB67**(*E_p*, *r=0.5*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.55991073}*)

Defines *ARIB STD-B67 (Hybrid Log-Gamma)* inverse opto-electrical transfer function (OETF / OECF).

Parameters

- **E_p** (numeric or array_like) – Non-linear signal E'.
- **r** (numeric, optional) – Video level corresponding to reference white level.
- **constants** (Structure, optional) – *ARIB STD-B67 (Hybrid Log-Gamma)* constants.

Returns Voltage E normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera color channel R, G, B.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

- This definition uses the *mirror* negative number handling mode of colour.models.gamma_function() definition to the sign of negative numbers.

References

[AssociationoRIaBusinesses15]

Examples

```
>>> oetf_inverse_ARIBSTDB67(0.212132034355964)
0.1799999...
```

colour.models.oetf_BT2020

colour.models.**oetf_BT2020**(*E*, *is_12_bits_system=False*, *constants={'alpha': <function <lambda>>, 'beta': <function <lambda>>}*)

Defines *Recommendation ITU-R BT.2020* opto-electrical transfer function (OETF / OECF).

Parameters

- **E** (numeric or array_like) – Voltage E normalised by the reference white level and proportional to the implicit light intensity that would be detected with a reference camera colour channel R, G, B.
- **is_12_bits_system** (bool, optional) – *BT.709 alpha* and *beta* constants are used if system is not 12-bit.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2020* constants.

Returns Resulting non-linear signal E'.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[InternationalTUnion15a]

Examples

```
>>> oetf_BT2020(0.18)
0.4090077...
```

colour.models.oetf_HLG_BT2100

colour.models.**oetf_HLG_BT2100**(*E*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-electrical transfer function (OETF / OECF).

The OETF maps relative scene linear light into the non-linear *HLG* signal value.

Parameters

- **E** (numeric or array_like) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

Returns E' is the resulting non-linear signal R', G', B'.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_HLG_BT2100(0.18 / 12)
0.2121320...
```

colour.models.oetf_inverse_HLG_BT2100

colour.models.**oetf_inverse_HLG_BT2100**(*E_p*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-electrical transfer function (OETF / OECF).

Parameters

- **E_p** (numeric or array_like) – E' is the resulting non-linear signal R', G', B'.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_inverse_HLG_BT2100(0.212132034355964)
0.0149999...
```

colour.models.oetf_PQ_BT2100

colour.models.**oetf_PQ_BT2100**(E)

Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-electrical transfer function (OETF / OECF).

The OETF maps relative scene linear light into the non-linear *PQ* signal value.

Parameters **E** (numeric or array_like) – $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Returns E' is the resulting non-linear signal (R', G', B').

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_PQ_BT2100(0.1)
0.7247698...
```

colour.models.oetf_inverse_PQ_BT2100

colour.models.**oetf_inverse_PQ_BT2100**(E_p)

Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-electrical transfer function (OETF / OECF).

Parameters **E_p** (numeric or array_like) – E' is the resulting non-linear signal (R', G', B').

Returns $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> oetf_inverse_PQ_BT2100(0.724769816665726)
0.0999999...
```

colour.models.oetf_BT601

colour.models.**oetf_BT601**(*L*)

Defines *Recommendation ITU-R BT.601-7* opto-electronic transfer function (OETF / OECF).

Parameters **L** (numeric or array_like) – *Luminance* L of the image.

Returns Corresponding electrical signal E.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[InternationalTUnion11b]

Examples

```
>>> oetf_BT601(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT601

colour.models.**oetf_inverse_BT601**(*E*)

Defines *Recommendation ITU-R BT.601-7* inverse opto-electronic transfer function (OETF / OECF).

Parameters **E** (numeric or array_like) – Electrical signal E.

Returns Corresponding *luminance* L of the image.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion11b]

Examples

```
>>> oetf_inverse_BT601(0.409007728864150)
0.1...
```

colour.models.oetf_BT709

colour.models.**oetf_BT709**(*L*)

Defines *Recommendation ITU-R BT.709-6* opto-electronic transfer function (OETF / OECF).

Parameters **L** (numeric or array_like) – *Luminance* L of the image.

Returns Corresponding electrical signal V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalTUnion15b]

Examples

```
>>> oetf_BT709(0.18)
0.4090077...
```

colour.models.oetf_inverse_BT709

colour.models.**oetf_inverse_BT709**(*V*)

Defines *Recommendation ITU-R BT.709-6* inverse opto-electronic transfer function (OETF / OECF).

Parameters **V** (numeric or array_like) – Electrical signal V.

Returns Corresponding *luminance* L of the image.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion15b]

Examples

```
>>> oetf_inverse_BT709(0.409007728864150)
0.1...
```

colour.models.oetf_SMPTE240M

colour.models.**oetf_SMPTE240M**(*L_c*)

Defines *SMPTE 240M* opto-electrical transfer function (OETF / OECF).

Parameters **L_c** (numeric or array_like) – Light input L_c to the reference camera normalised to the system reference white.

Returns Video signal output V_c of the reference camera normalised to the system reference white.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V_c	[0, 1]	[0, 1]

References

[SocietyoMPaTEngineers99]

Examples

```
>>> oetf_SMPTE240M(0.18)
0.4022857...
```

Ancillary Objects

colour

gamma_function(a[, exponent, ...])	Defines a typical gamma encoding / decoding function.
linear_function(a)	Defines a typical linear encoding / decoding function, essentially a pass-through function.

colour.gamma_function

colour.**gamma_function**(*a*, *exponent=1*, *negative_number_handling='Indeterminate'*)

Defines a typical gamma encoding / decoding function.

Parameters

- **a** (numeric or array_like) – Array to encode / decode.
- **exponent** (numeric or array_like, optional) – Encoding / decoding exponent.
- **negative_number_handling** (unicode, optional) – **{'Indeterminate', 'Mirror', 'Preserve', 'Clamp'}**, Defines the behaviour for a negative numbers and / or the definition return value:
  - *Indeterminate*: The behaviour will be indeterminate and definition return value might contain *nans*.
  - *Mirror*: The definition return value will be mirrored around abscissa and ordinate axis, i.e. Blackmagic Design: Davinci Resolve behaviour.
  - *Preserve*: The definition will preserve any negative number in a, i.e. The Foundry Nuke behaviour.
  - *Clamp*: The definition will clamp any negative number in a to 0.

Returns Encoded / decoded array.

Return type numeric or ndarray

Raises ValueError – If the negative number handling method is not defined.

Examples

```
>>> gamma_function(0.18, 2.2)
0.0229932...
>>> gamma_function(-0.18, 2.0)
0.0323999...
>>> gamma_function(-0.18, 2.2)
nan
>>> gamma_function(-0.18, 2.2, 'Mirror')
-0.0229932...
>>> gamma_function(-0.18, 2.2, 'Preserve')
-0.1...
>>> gamma_function(-0.18, 2.2, 'Clamp')
0.0
```

colour.linear_function

colour.**linear_function**(*a*)

Defines a typical linear encoding / decoding function, essentially a pass-through function.

Parameters **a** (numeric or array_like) – Array to encode / decode.

Returns Encoded / decoded array.

Return type numeric or ndarray

Examples

```
>>> linear_function(0.18)
0.18
```

Electro-Optical Transfer Functions

colour

eotf(value[, function])	Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF / EOCF).
EOTFS	Supported electro-optical transfer functions (EOTFs / EOCFs).
eotf_inverse(value[, function])	Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given inverse electro-optical transfer function (EOTF / EOCF).
EOTF_INVERSES	Supported inverse electro-optical transfer functions (EOTFs / EOCFs).

colour.eotf

colour.**eotf**(*value*, *function='ITU-R BT.1886'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to tristimulus values at the display using given electro-optical transfer function (EOTF / EOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.1886', 'DCDM', 'DICOM GSDF', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'SMPTE 240M', 'ST 2084', 'sRGB'}**, Electro-optical transfer function (EOTF / EOCF).

Other Parameters

- **E_clip** (*numeric, optional*) – {colour.models.cctf_decoding_RIMMRGB()}, Maximum exposure level.
- **I_max** (*numeric, optional*) – {colour.models.cctf_decoding_ROMMRGB(), colour.models.cctf_decoding_RIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **L_B** (*numeric, optional*) – {colour.models.eotf_BT1886(), colour.models.eotf_HLG_BT2100()}, Screen luminance for black.
- **L_W** (*numeric, optional*) – {colour.models.eotf_BT1886(), colour.models.eotf_HLG_BT2100()}, Screen luminance for white.
- **L_p** (*numeric, optional*) – {colour.models.eotf_ST2084()}, Display peak luminance cd/m^2.
- **gamma** (*numeric, optional*) – {colour.models.eotf_HLG_BT2100()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **is_12_bits_system** (*bool, optional*) – {colour.models.eotf_BT2020()}, *ITU-R BT.2020 alpha* and *beta* constants are used if system is not 12-bit.
- **method** (*unicode, optional*) – {colour.models.eotf_HLG_BT2100()}, **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**
- **out_int** (*bool, optional*) – {colour.models.eotf_DCDM(), colour.models.eotf_DICOMGSDF()}, Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Tristimulus values at the display.

Return type numeric or ndarray

Examples

```
>>> eotf(0.461356129500442)
0.1...
>>> eotf(0.409007728864150, function='ITU-R BT.2020')
...
0.1...
>>> eotf(0.182011532850008, function='ST 2084', L_p=1000)
...
0.1...
```

colour.EOTFS

colour.**EOTFS = CaseInsensitiveMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU-R BT.20**
Supported electro-optical transfer functions (EOTFs / EOCFs).

EOTFS [CaseInsensitiveMapping] **{'DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2020', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'SMPTE 240M', 'ST 2084', 'sRGB'}**

colour.eotf_inverse

colour.**eotf_inverse**(*value*, *function='ITU-R BT.1886'*, ***kwargs*)
Encodes estimated tristimulus values in a scene to $R'G'B'$ video component signal value using given inverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.1886', 'DCDM', 'DICOM GSDF', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ST 2084', 'sRGB'}**, Inverse electro-optical transfer function (EOTF / EOCF).

Other Parameters

- **L_B** (*numeric, optional*) – {colour.models.eotf_inverse_BT1886(), colour.models.eotf_inverse_HLG_BT2100()}, Screen luminance for black.
- **L_W** (*numeric, optional*) – {colour.models.eotf_inverse_BT1886(), colour.models.eotf_inverse_HLG_BT2100()}, Screen luminance for white.
- **gamma** (*numeric, optional*) – {colour.models.eotf_HLG_BT2100()}, System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **L_p** (*numeric, optional*) – {colour.models.eotf_inverse_ST2084()}, Display peak luminance cd/m^2.
- **method** (*unicode, optional*) – {colour.models.eotf_inverse_HLG_BT2100()}, **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**
- **out_int** (*bool, optional*) – {colour.models.eotf_inverse_DCDM(), colour.models.eotf_inverse_DICOMGSDF()}, Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns $R'G'B'$ video component signal value.

Return type numeric or ndarray

Examples

```
>>> eotf_inverse(0.11699185725296059)
0.4090077...
>>> eotf_inverse(
...     0.11699185725296059, function='ITU-R BT.1886')
0.4090077...
```

colour.EOTF_INVERSES

colour.**EOTF_INVERSES = CaseInsensitiveMapping({'DCDM': ..., 'DICOM GSDF': ..., 'ITU-R BT.1886': ..., 'ITU**
Supported inverse electro-optical transfer functions (EOTFs / EOCFs).

EOTF_INVERSES [CaseInsensitiveMapping] **{'DCDM', 'DICOM GSDF', 'ITU-R BT.1886', 'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ', 'ST 2084', 'sRGB'}**

colour.models

eotf_DCDM(XYZ_p[, in_int])	Defines the *DCDM* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_DCDM(XYZ[, out_int])	Defines the *DCDM* inverse electro-optical transfer function (EOTF / EOCF).
eotf_DICOMGSDF(J[, in_int])	Defines the *DICOM - Grayscale Standard Display Function* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_DICOMGSDF(L[, out_int])	Defines the *DICOM - Grayscale Standard Display Function* inverse electro-optical transfer function (EOTF / EOCF).
eotf_BT1886(V[, L_B, L_W])	Defines *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_BT1886(L[, L_B, L_W])	Defines *Recommendation ITU-R BT.1886* inverse electro-optical transfer function (EOTF / EOCF).
eotf_BT2020(E_p[, is_12_bits_system, constants])	Defines *Recommendation ITU-R BT.2020* electro-optical transfer function (EOTF / EOCF).
BT2100_HLG_EOTF_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).
eotf_HLG_BT2100(E_p[, L_B, L_W, gamma, ...])	Defines *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).
BT2100_HLG_EOTF_INVERSE_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).
eotf_inverse_HLG_BT2100(F_D[, L_B, L_W, ...])	Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).
eotf_PQ_BT2100(E_p)	Defines *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_PQ_BT2100(F_D)	Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse electro-optical transfer function (EOTF / EOCF).
eotf_SMPTE240M(V_r)	Defines *SMPTE 240M* electro-optical transfer function (EOTF / EOCF).
eotf_ST2084(N[, L_p, constants])	Defines *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF / EOCF).
eotf_inverse_ST2084(C[, L_p, constants])	Defines *SMPTE ST 2084:2014* optimised perceptual inverse electro-optical transfer function (EOTF / EOCF).
eotf_sRGB(V)	Defines the *IEC 61966-2-1:1999 sRGB* electro-optical transfer function (EOTF / EOCF).
eotf_inverse_sRGB(L)	Defines the *IEC 61966-2-1:1999 sRGB* inverse electro-optical transfer function (EOTF / EOCF).

colour.models.eotf_DCDM

colour.models.**eotf_DCDM**(*XYZ_p*, *in_int=False*)

Defines the *DCDM* electro-optical transfer function (EOTF / EOCF).

Parameters

- **XYZ_p** (numeric or array_like) – Non-linear *CIE XYZ'* tristimulus values.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns *CIE XYZ* tristimulus values.

Return type numeric or ndarray

Warning: *DCDM* is an absolute transfer function.

Notes

- *DCDM* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain *	Scale - Reference	Scale - 1
XYZ_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[DigitalCInitiatives07]

Examples

```
>>> eotf_DCDM(0.11281860951766724)
0.18...
>>> eotf_DCDM(462, in_int=True)
0.18...
```

colour.models.eotf_inverse_DCDM

colour.models.**eotf_inverse_DCDM**(*XYZ*, *out_int=False*)

Defines the *DCDM* inverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **XYZ** (numeric or array_like) – *CIE XYZ* tristimulus values.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Non-linear *CIE XYZ*' tristimulus values.

Return type numeric or ndarray

Warning: *DCDM* is an absolute transfer function.

Notes

- *DCDM* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain *	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
XYZ_p	[0, 1]	[0, 1]

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[DigitalCInitiatives07]

Examples

```
>>> eotf_inverse_DCDM(0.18)
0.1128186...
>>> eotf_inverse_DCDM(0.18, out_int=True)
462
```

colour.models.eotf_DICOMGSDF

colour.models.**eotf_DICOMGSDF**(*J*, *in_int=False*)

Defines the *DICOM - Grayscale Standard Display Function* electro-optical transfer function (EOTF / EOCF).

Parameters

- **J** (numeric or array_like) – Just-Noticeable Difference (JND) Index, j.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.

Returns Corresponding *luminance* L.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
J	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[NationalEMAssociation04]

Examples

```
>>> eotf_DICOMGSDF(0.500486263438448)
130.0628647...
>>> eotf_DICOMGSDF(512, in_int=True)
130.0652840...
```

colour.models.eotf_inverse_DICOMGSDF

colour.models.**eotf_inverse_DICOMGSDF**(*L*, *out_int=False*)

Defines the *DICOM - Grayscale Standard Display Function* inverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **L** (numeric or array_like) – *Luminance* L.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Just-Noticeable Difference (JND) Index, j.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
J	[0, 1]	[0, 1]

References

[NationalEMAssociation04]

Examples

```
>>> eotf_inverse_DICOMGSDF(130.0662)
0.5004862...
>>> eotf_inverse_DICOMGSDF(130.0662, out_int=True)
512
```

colour.models.eotf_BT1886

colour.models.**eotf_BT1886**(*V*, *L_B=0*, *L_W=1*)

Defines *Recommendation ITU-R BT.1886* electro-optical transfer function (EOTF / EOCF).

Parameters

- **V** (numeric or array_like) – Input video signal level (normalised, black at $V = 0$, to white at $V = 1$. For content mastered per *Recommendation ITU-R BT.709*, 10-bit digital code values D map into values of V per the following equation: $V = (D - 64)/876$
- **L_B** (numeric, optional) – Screen luminance for black.
- **L_W** (numeric, optional) – Screen luminance for white.

Returns Screen luminance in cd/m^2.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalTUnion11a]

Examples

```
>>> eotf_BT1886(0.409007728864150)
0.1169918...
```

colour.models.eotf_inverse_BT1886

colour.models.**eotf_inverse_BT1886**(*L*, *L_B=0*, *L_W=1*)

Defines *Recommendation ITU-R BT.1886* inverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **L** (numeric or array_like) – Screen luminance in cd/m^2.
- **L_B** (numeric, optional) – Screen luminance for black.
- **L_W** (numeric, optional) – Screen luminance for white.

Returns Input video signal level (normalised, black at $V = 0$, to white at $V = 1$.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalTUnion11a]

Examples

```
>>> eotf_inverse_BT1886(0.11699185725296059)
0.4090077...
```

colour.models.eotf_BT2020

colour.models.**eotf_BT2020**(*E_p*, *is_12_bits_system=False*, *constants={'alpha': <function <lambda>>, 'beta': <function <lambda>>}*)

Defines *Recommendation ITU-R BT.2020* electro-optical transfer function (EOTF / EOCF).

Parameters

- **E_p** (numeric or array_like) – Non-linear signal E'.
- **is_12_bits_system** (bool, optional) – *BT.709 alpha* and *beta* constants are used if system is not 12-bit.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2020* constants.

Returns Resulting voltage E.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[InternationalTUnion15a]

Examples

```
>>> eotf_BT2020(0.705515089922121)
0.4999999...
```

colour.models.BT2100_HLG_EOTF_METHODS

colour.models.**BT2100_HLG_EOTF_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2**

Supported *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

BT2100_HLG_EOTF_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.eotf_HLG_BT2100

colour.models.**eotf_HLG_BT2100**(*E_p*, *L_B=0*, *L_W=1000*, *gamma=None*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*, *method='ITU-R BT.2100-2'*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* electro-optical transfer function (EOTF / EOCF).

The EOTF maps the non-linear *HLG* signal into display light.

Parameters

- **E_p** (numeric or array_like) – E' denotes a non-linear colour value R', G', B' or L', M', S' in *HLG* space.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000 cd/m^2$.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.

- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

Returns Luminance F_D of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

Examples

```
>>> eotf_HLG_BT2100(0.212132034355964)
6.4760398...
>>> eotf_HLG_BT2100(0.212132034355964, method='ITU-R BT.2100-1')
...
6.4760398...
>>> eotf_HLG_BT2100(0.212132034355964, 0.01)
...
7.3321975...
```

colour.models.BT2100_HLG_EOTF_INVERSE_METHODS

colour.models.**BT2100_HLG_EOTF_INVERSE_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT**

Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

BT2100_HLG_EOTF_INVERSE_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.eotf_inverse_HLG_BT2100

colour.models.**eotf_inverse_HLG_BT2100**(*F_D*, *L_B=0*, *L_W=1000*, *gamma=None*, *constants={'a': 0.17883277, 'b': 0.28466892, 'c': 0.559910729529562}*, *method='ITU-R BT.2100-2'*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **F_D** (numeric or array_like) – Luminance F_D of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **constants** (Structure, optional) – *Recommendation ITU-R BT.2100 Reference HLG* constants.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in *HLG* space.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

Examples

```
>>> eotf_inverse_HLG_BT2100(6.476039825649814)
0.2121320...
>>> eotf_inverse_HLG_BT2100(6.476039825649814, method='ITU-R BT.2100-1')
...
0.2121320...
>>> eotf_inverse_HLG_BT2100(7.332197528353875, 0.01)
0.2121320...
```

colour.models.eotf_PQ_BT2100

colour.models.**eotf_PQ_BT2100**(*E_p*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* electro-optical transfer function (EOTF / EOCF).

The EOTF maps the non-linear *PQ* signal into display light.

Parameters **E_p** (numeric or array_like) – E' denotes a non-linear colour value R', G', B' or L', M', S' in *PQ* space [0, 1].

Returns F_D is the luminance of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> eotf_PQ_BT2100(0.724769816665726)
779.9883608...
```

colour.models.eotf_inverse_PQ_BT2100

colour.models.**eotf_inverse_PQ_BT2100**(*F_D*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse electro-optical transfer function (EOTF / EOCF).

Parameters **F_D** (numeric or array_like) – F_D is the luminance of a displayed linear component R_D, G_D, B_D or Y_D or I_D, in cd/m^2.

Returns E' denotes a non-linear colour value R', G', B' or L', M', S' in *PQ* space [0, 1].

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E_p	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> eotf_inverse_PQ_BT2100(779.988360834085370)
0.7247698...
```

colour.models.eotf_SMPTE240M

colour.models.**eotf_SMPTE240M**(*V_r*)

Defines *SMPTE 240M* electro-optical transfer function (EOTF / EOCF).

Parameters **V_r** (numeric or array_like) – Video signal level V_r driving the reference reproducer normalised to the system reference white.

Returns Light output L_r from the reference reproducer normalised to the system reference white.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V_c	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_c	[0, 1]	[0, 1]

References

[SocietyoMPaTEngineers99]

Examples

```
>>> eotf_SMPTE240M(0.402285796753870)
0.1...
```

colour.models.eotf_ST2084

colour.models.**eotf_ST2084**(*N*, *L_p=10000*, *constants={'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'm_1': 0.1593017578125, 'm_2': 78.84375}*)

Defines *SMPTE ST 2084:2014* optimised perceptual electro-optical transfer function (EOTF / EOCF).

This perceptual quantizer (PQ) has been modeled by Dolby Laboratories using *Barten (1999)* contrast sensitivity function.

Parameters

- **N** (numeric or array_like) – Color value abbreviated as N, that is directly proportional to the encoded signal representation, and which is not directly proportional to the optical output of a display device.
- **L_p** (numeric, optional) – System peak luminance cd/m^2, this parameter should stay at its default $10000cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.
- **constants** (Structure, optional) – *SMPTE ST 2084:2014* constants.

Returns Target optical output C in cd/m^2 of the ideal reference display.

Return type numeric or ndarray

Warning: *SMPTE ST 2084:2014* is an absolute transfer function.

Notes

- *SMPTE ST 2084:2014* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
N	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
C	[0, 1]	[0, 1]

References

[Mil14], [SocietyoMPaTEngineers14]

Examples

```
>>> eotf_ST2084(0.508078421517399)
100.0000000...
```

colour.models.eotf_inverse_ST2084

colour.models.**eotf_inverse_ST2084**(*C*, *L_p=10000*, *constants={'c_1': 0.8359375, 'c_2': 18.8515625, 'c_3': 18.6875, 'm_1': 0.1593017578125, 'm_2': 78.84375}*)

Defines *SMPTE ST 2084:2014* optimised perceptual inverse electro-optical transfer function (EOTF / EOCF).

Parameters

- **C** (numeric or array_like) – Target optical output C in cd/m^2 of the ideal reference display.
- **L_p** (numeric, optional) – System peak luminance cd/m^2, this parameter should stay at its default $10000 cd/m^2$ value for practical applications. It is exposed so that the definition can be used as a fitting function.
- **constants** (Structure, optional) – *SMPTE ST 2084:2014* constants.

Returns Color value abbreviated as N, that is directly proportional to the encoded signal representation, and which is not directly proportional to the optical output of a display device.

Return type numeric or ndarray

Warning: *SMPTE ST 2084:2014* is an absolute transfer function.

Notes

- *SMPTE ST 2084:2014* is an absolute transfer function, thus the domain and range values for the *Reference* and *1* scales are only indicative that the data is not affected by scale transformations.

Domain	Scale - Reference	Scale - 1
C	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
N	[0, 1]	[0, 1]

References

[Mil14], [SocietyoMPaTEngineers14]

Examples

```
>>> eotf_inverse_ST2084(100)
0.5080784...
```

colour.models.eotf_sRGB

colour.models.**eotf_sRGB**(*V*)

Defines the *IEC 61966-2-1:1999 sRGB* electro-optical transfer function (EOTF / EOCF).

Parameters **V** (numeric or array_like) – Electrical signal V.

Returns Corresponding *luminance* L of the image.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

References

[InternationalECommission99], [InternationalTUnion15b]

Examples

```
>>> eotf_sRGB(0.461356129500442)
0.1...
```

colour.models.eotf_inverse_sRGB

colour.models.**eotf_inverse_sRGB**(*L*)

Defines the *IEC 61966-2-1:1999 sRGB* inverse electro-optical transfer function (EOTF / EOCF).

Parameters **L** (numeric or array_like) – *Luminance* L of the image.

Returns Corresponding electrical signal V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 1]	[0, 1]

References

[InternationalECommission99], [InternationalTUnion15b]

Examples

```
>>> eotf_inverse_sRGB(0.18)
0.4613561...
```

Opto-Optical Transfer Functions

colour

ootf(value[, function])	Maps relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).
OOTFS	Supported opto-optical transfer functions (OOTFs / OOCFs).
ootf_inverse(value[, function])	Maps relative display linear light to scene linear light using given inverse opto-optical transfer function (OOTF / OOCF).
OOTF_INVERSES	Supported inverse opto-optical transfer functions (OOTFs / OOCFs).

colour.ootf

colour.**ootf**(*value*, *function='ITU-R BT.2100 PQ'*, ***kwargs*)

Maps relative scene linear light to display linear light using given opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}** Opto-optical transfer function (OOTF / OOCF).

Returns Luminance of a displayed linear component.

Return type numeric or ndarray

Examples

```
>>> ootf(0.1)
779.9883608...
>>> ootf(0.1, function='ITU-R BT.2100 HLG')
63.0957344...
```

colour.OOTFS

colour.**OOTFS = CaseInsensitiveMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})**

Supported opto-optical transfer functions (OOTFs / OOCFs).

OOTFS [CaseInsensitiveMapping] **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**

colour.ootf_inverse

colour.**ootf_inverse**(*value*, *function='ITU-R BT.2100 PQ'*, ***kwargs*)

Maps relative display linear light to scene linear light using given inverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}** Inverse opto-optical transfer function (OOTF / OOCF).

Other Parameters

- **L_B** (*numeric, optional*) – {colour.models.ootf_inverse_HLG_BT2100()}, L_B is the display luminance for black in cd/m^2.
- **L_W** (*numeric, optional*) – {colour.models.ootf_inverse_HLG_BT2100()}, L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (*numeric, optional*) – {colour.models.ootf_inverse_HLG_BT2100()}, System gamma value, 1.2 at the nominal display peak luminance of $1000 cd/m^2$.

Returns Luminance of scene linear light.

Return type numeric or ndarray

Examples

```
>>> ootf_inverse(779.988360834115840)
0.1000000...
>>> ootf_inverse(
...     63.095734448019336, function='ITU-R BT.2100 HLG')
0.1000000...
```

colour.OOTF_INVERSES

colour.**OOTF_INVERSES = CaseInsensitiveMapping({'ITU-R BT.2100 HLG': ..., 'ITU-R BT.2100 PQ': ...})**
Supported inverse opto-optical transfer functions (OOTFs / OOCFs).

OOTF_INVERSES [CaseInsensitiveMapping] **{'ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ'}**

colour.models

BT2100_HLG_OOTF_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
ootf_HLG_BT2100(E[, L_B, L_W, gamma, method])	Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).
BT2100_HLG_OOTF_INVERSE_METHODS	Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).
ootf_inverse_HLG_BT2100(F_D[, L_B, L_W, ...])	Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).
ootf_PQ_BT2100(E)	Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).
ootf_inverse_PQ_BT2100(F_D)	Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-optical transfer function (OOTF / OOCF).

colour.models.BT2100_HLG_OOTF_METHODS

colour.models.**BT2100_HLG_OOTF_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT.2100-2**
Supported *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

BT2100_HLG_OOTF_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.ootf_HLG_BT2100

colour.models.**ootf_HLG_BT2100**(*E*, *L_B=0*, *L_W=1000*, *gamma=None*, *method='ITU-R BT.2100-2'*)
Defines *Recommendation ITU-R BT.2100 Reference HLG* opto-optical transfer function (OOTF / OOCF).

The OOTF maps relative scene linear light to display linear light.

Parameters

- **E** (numeric or array_like) – E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.

- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

Returns F_D is the luminance of a displayed linear component R_D, G_D, orB_D, in cd/m^2.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> ootf_HLG_BT2100(0.1)
63.0957344...
>>> ootf_HLG_BT2100(0.1, 0.01, method='ITU-R BT.2100-1')
...
63.1051034...
```

colour.models.BT2100_HLG_OOTF_INVERSE_METHODS

colour.models.**BT2100_HLG_OOTF_INVERSE_METHODS = CaseInsensitiveMapping({'ITU-R BT.2100-1': ..., 'ITU-R BT**

Supported *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

BT2100_HLG_OOTF_INVERSE_METHODS [CaseInsensitiveMapping] **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**

colour.models.ootf_inverse_HLG_BT2100

colour.models.**ootf_inverse_HLG_BT2100**(*F_D*, *L_B=0*, *L_W=1000*, *gamma=None*, *method='ITU-R BT.2100-2'*)

Defines *Recommendation ITU-R BT.2100 Reference HLG* inverse opto-optical transfer function (OOTF / OOCF).

Parameters

- **F_D** (numeric or array_like) – F_D is the luminance of a displayed linear component $R_D, G_D, or B_D$, in cd/m^2.
- **L_B** (numeric, optional) – L_B is the display luminance for black in cd/m^2.
- **L_W** (numeric, optional) – L_W is nominal peak luminance of the display in cd/m^2 for achromatic pixels.
- **gamma** (numeric, optional) – System gamma value, 1.2 at the nominal display peak luminance of $1000cd/m^2$.
- **method** (unicode, optional) – **{'ITU-R BT.2100-1', 'ITU-R BT.2100-2'}**, Computation method.

Returns E is the signal for each colour component R_S, G_S, B_S proportional to scene linear light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17], [InternationalTUnion18]

Examples

```
>>> ootf_inverse_HLG_BT2100(63.095734448019336)
0.1000000...
>>> ootf_inverse_HLG_BT2100(
...     63.105103490674857, 0.01, method='ITU-R BT.2100-1')
...
0.0999999...
```

colour.models.ootf_PQ_BT2100

colour.models.**ootf_PQ_BT2100**(*E*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* opto-optical transfer function (OOTF / OOCF).

The OOTF maps relative scene linear light to display linear light.

Parameters **E** (numeric or array_like) – $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Returns F_D is the luminance of a displayed linear component (R_D, G_D, B_D; Y_D; or I_D).

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> ootf_PQ_BT2100(0.1)
779.9883608...
```

colour.models.ootf_inverse_PQ_BT2100

colour.models.**ootf_inverse_PQ_BT2100**(*F_D*)

Defines *Recommendation ITU-R BT.2100 Reference PQ* inverse opto-optical transfer function (OOTF / OOCF).

Parameters **F_D** (numeric or array_like) – F_D is the luminance of a displayed linear component (R_D, G_D, B_D; Y_D; or I_D).

Returns $E = R_S, G_S, B_S; Y_S; or I_S$ is the signal determined by scene light and scaled by camera exposure.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
F_D	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
E	[0, 1]	[0, 1]

References

[Bor17], [InternationalTUnion17]

Examples

```
>>> ootf_inverse_PQ_BT2100(779.988360834115840)
0.1000000...
```

Log Encoding and Decoding

colour

log_encoding(value[, function])	Encodes linear-light values to $R'G'B'$ video component signal value using given *log* function.
LOG_ENCODINGS	Supported *log* encoding functions.
log_decoding(value[, function])	Decodes $R'G'B'$ video component signal value to linear-light values using given *log* function.
LOG_DECODINGS	Supported *log* decoding functions.

colour.log_encoding

colour.**log_encoding**(*value*, *function='Cineon'*, ***kwargs*)

Encodes linear-light values to $R'G'B'$ video component signal value using given *log* function.

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**, Computation function.

Other Parameters

- **EI** (*int, optional*) – {colour.models.log_encoding_ALEXALogC()}, Ei.
- **E_clip** (*numeric, optional*) – {colour.models.log_encoding_ERIMMRGB()}, Maximum exposure limit.
- **E_min** (*numeric, optional*) – {colour.models.log_encoding_ERIMMRGB()}, Minimum exposure limit.
- **I_max** (*numeric, optional*) – {colour.models.log_encoding_ERIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit

per channel.

- **bit_depth** (*unicode, optional*) – {colour.models.log_encoding_ACESproxy(), colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2()}, **{8, 10, 12}**, Bit depth used for conversion, *ACESproxy* uses **{10, 12}**.
- **black_offset** (*numeric or array_like*) – {colour.models.log_encoding_Cineon(), colour.models.log_encoding_Panalog(), colour.models.log_encoding_REDLog(), colour.models.log_encoding_REDLogFilm()}, Black offset.
- **density_per_code_value** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Density per code value.
- **firmware** (*unicode, optional*) – {colour.models.log_encoding_ALEXALogC()}, **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **in_reflection** (*bool, optional*) – {colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2()}, Whether the light level x to a camera is reflection.
- **linear_reference** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Linear reference.
- **log_reference** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Log reference.
- **method** (*unicode, optional*) – {colour.models.log_encoding_Log3G10()}, Whether to use the *Log3G10 v1* or *v2* log encoding curve.
- **out_normalised_code_value** (*bool, optional*) – {colour.models.log_encoding_SLog(), colour.models.log_encoding_SLog2(), colour.models.log_encoding_SLog3()}, Whether the non-linear *Sony S-Log*, *Sony S-Log2* or *Sony S-Log3* data y is encoded as normalised code values.
- **negative_gamma** (*numeric or array_like*) – {colour.models.log_encoding_PivotedLog()}, Negative gamma.
- **method** (*unicode, optional*) – {colour.models.log_encoding_ALEXALogC()}, **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.

Returns *Log* value.

Return type numeric or ndarray

Examples

```
>>> log_encoding(0.18)
0.4573196...
>>> log_encoding(0.18, function='ACEScc')
0.4135884...
>>> log_encoding(0.18, function='PLog', log_reference=400)
...
0.3910068...
>>> log_encoding(0.18, function='S-Log')
0.3849708...
```

colour.LOG_ENCODINGS

colour.**LOG_ENCODINGS = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA**

Supported *log* encoding functions.

LOG_ENCODINGS [CaseInsensitiveMapping] **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**

colour.log_decoding

colour.**log_decoding**(*value*, *function='Cineon'*, ***kwargs*)

Decodes $R'G'B'$ video component signal value to linear-light values using given *log* function.

Parameters

- **value** (numeric or array_like) – Value.
- **function** (unicode, optional) – **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**, Computation function.

Other Parameters

- **EI** (*int, optional*) – {colour.models.log_decoding_ALEXALogC()}, Ei.
- **E_clip** (*numeric, optional*) – {colour.models.log_decoding_ERIMMRGB()}, Maximum exposure limit.
- **E_min** (*numeric, optional*) – {colour.models.log_decoding_ERIMMRGB()}, Minimum exposure limit.
- **I_max** (*numeric, optional*) – {colour.models.log_decoding_ERIMMRGB()}, Maximum code value: 255, 4095 and 650535 for respectively 8-bit, 12-bit and 16-bit per channel.
- **bit_depth** (*int, optional*) – {colour.models.log_decoding_ACESproxy(), colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2()}, **{8, 10, 12}**, Bit depth used for conversion, *ACESproxy* uses **{10, 12}**.
- **black_offset** (*numeric or array_like*) – {colour.models.log_decoding_Cineon(), colour.models.log_decoding_Panalog(), colour.models.log_decoding_REDLog(), colour.models.log_decoding_REDLogFilm()}, Black offset.
- **density_per_code_value** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Density per code value.
- **firmware** (*unicode, optional*) – {colour.models.log_decoding_ALEXALogC()}, **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **in_normalised_code_value** (*bool, optional*) – {colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2(), colour.models.log_decoding_SLog3()}, Whether the non-linear *Sony S-Log*, *Sony S-Log2* or *Sony S-Log3* data y is encoded as normalised code values.
- **linear_reference** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Linear reference.
- **log_reference** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Log reference.

- **method** (*unicode, optional*) – {colour.models.log_decoding_Log3G10()}, Whether to use the *Log3G10 v1* or *v2* log encoding curve.
- **negative_gamma** (*numeric or array_like*) – {colour.models.log_decoding_PivotedLog()}, Negative gamma.
- **out_reflection** (*bool, optional*) – {colour.models.log_decoding_SLog(), colour.models.log_decoding_SLog2()}, Whether the light level x to a camera is reflection.
- **method** (*unicode, optional*) – {colour.models.log_decoding_ALEXALogC()}, **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.

Returns *Log* value.

Return type numeric or ndarray

Examples

```
>>> log_decoding(0.457319613085418)
0.1...
>>> log_decoding(0.413588402492442, function='ACEScc')
...
0.1...
>>> log_decoding(0.391006842619746, function='PLog', log_reference=400)
...
0.1...
>>> log_decoding(0.376512722254600, function='S-Log')
...
0.1...
```

colour.LOG_DECODINGS

colour.**LOG_DECODINGS = CaseInsensitiveMapping({'ACEScc': ..., 'ACEScct': ..., 'ACESproxy': ..., 'ALEXA**

Supported *log* decoding functions.

LOG_DECODINGS [CaseInsensitiveMapping] **{'ACEScc', 'ACEScct', 'ACESproxy', 'ALEXA Log C', 'Canon Log 2', 'Canon Log 3', 'Canon Log', 'Cineon', 'D-Log', 'ERIMM RGB', 'F-Log', 'Filmic Pro 6', 'Log3G10', 'Log3G12', 'Panalog', 'PLog', 'Protune', 'REDLog', 'REDLogFilm', 'S-Log', 'S-Log2', 'S-Log3', 'T-Log', 'V-Log', 'ViperLog'}**

colour.models

log_encoding_ACEScc(lin_AP1)	Defines the *ACEScc* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScc(ACEScc)	Defines the *ACEScc* colourspace log decoding / electro-optical transfer function.
log_encoding_ACEScct(lin_AP1[, constants])	Defines the *ACEScct* colourspace log encoding / opto-electronic transfer function.
log_decoding_ACEScct(ACEScct[, constants])	Defines the *ACEScct* colourspace log decoding / electro-optical transfer function.
log_encoding_ACESproxy(lin_AP1[, bit_depth, ...])	Defines the *ACESproxy* colourspace log encoding curve / opto-electronic transfer function.
log_decoding_ACESproxy(ACESproxy[, ...])	Defines the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.
log_encoding_ALEXALogC(x[, firmware, method, EI])	Defines the *ARRI ALEXA Log C* log encoding curve / opto-electronic transfer function.

Continued on next page

Table 180 – continued from previous page

log_decoding_ALEXALogC(t[, firmware, method, EI])	Defines the *ARRI ALEXA Log C* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog2(x[, bit_depth, ...])	Defines the *Canon Log 2* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog2(clog2[, bit_depth, ...])	Defines the *Canon Log 2* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog3(x[, bit_depth, ...])	Defines the *Canon Log 3* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog3(clog3[, bit_depth, ...])	Defines the *Canon Log 3* log decoding curve / electro-optical transfer function.
log_encoding_CanonLog(x[, bit_depth, ...])	Defines the *Canon Log* log encoding curve / opto-electronic transfer function.
log_decoding_CanonLog(clog[, bit_depth, ...])	Defines the *Canon Log* log decoding curve / electro-optical transfer function.
log_encoding_Cineon(x[, black_offset])	Defines the *Cineon* log encoding curve / opto-electronic transfer function.
log_decoding_Cineon(y[, black_offset])	Defines the *Cineon* log decoding curve / electro-optical transfer function.
log_encoding_ERIMMRGB(X[, bit_depth, ...])	Defines the *ERIMM RGB* log encoding curve / opto-electronic transfer function (OETF / OECF).
log_decoding_ERIMMRGB(X_p[, bit_depth, ...])	Defines the *ERIMM RGB* log decoding curve / electro-optical transfer function (EOTF / EOCF).
log_encoding_FLog(in_r[, bit_depth, ...])	Defines the *Fujifilm F-Log* log encoding curve / opto-electronic transfer function.
log_decoding_FLog(out_r[, bit_depth, ...])	Defines the *Fujifilm F-Log* log decoding curve / electro-optical transfer function.
LOG3G10_ENCODING_METHODS	Supported *Log3G10* log encoding curve / opto-electronic transfer function methods.
log_encoding_Log3G10(x[, method])	Defines the *Log3G10* log encoding curve / opto-electronic transfer function.
LOG3G10_DECODING_METHODS	Supported *Log3G10* log decoding curve / electro-optical transfer function methods.
log_decoding_Log3G10(y[, method])	Defines the *Log3G10* log decoding curve / electro-optical transfer function.
log_encoding_Log3G12(x)	Defines the *Log3G12* log encoding curve / opto-electronic transfer function.
log_decoding_Log3G12(y)	Defines the *Log3G12* log decoding curve / electro-optical transfer function.
log_encoding_Panalog(x[, black_offset])	Defines the *Panalog* log encoding curve / opto-electronic transfer function.
log_decoding_Panalog(y[, black_offset])	Defines the *Panalog* log decoding curve / electro-optical transfer function.
log_encoding_PivotedLog(x[, log_reference, ...])	Defines the *Josh Pines* style *Pivoted Log* log encoding curve / opto-electronic transfer function.
log_decoding_PivotedLog(y[, log_reference, ...])	Defines the *Josh Pines* style *Pivoted Log* log decoding curve / electro-optical transfer function.
log_encoding_Protune(x)	Defines the *Protune* log encoding curve / opto-electronic transfer function.
log_decoding_Protune(y)	Defines the *Protune* log decoding curve / electro-optical transfer function.
log_encoding_REDLog(x[, black_offset])	Defines the *REDLog* log encoding curve / opto-electronic transfer function.
log_decoding_REDLog(y[, black_offset])	Defines the *REDLog* log decoding curve / electro-optical transfer function.

Continued on next page

Table 180 – continued from previous page

log_encoding_REDLogFilm(x[, black_offset])	Defines the *REDLogFilm* log encoding curve / opto-electronic transfer function.
log_decoding_REDLogFilm(y[, black_offset])	Defines the *REDLogFilm* log decoding curve / electro-optical transfer function.
log_encoding_SLog(x[, bit_depth, ...])	Defines the *Sony S-Log* log encoding curve / opto-electronic transfer function.
log_decoding_SLog(y[, bit_depth, ...])	Defines the *Sony S-Log* log decoding curve / electro-optical transfer function.
log_encoding_SLog2(x[, bit_depth, ...])	Defines the *Sony S-Log2* log encoding curve / opto-electronic transfer function.
log_decoding_SLog2(y[, bit_depth, ...])	Defines the *Sony S-Log2* log decoding curve / electro-optical transfer function.
log_encoding_SLog3(x[, bit_depth, ...])	Defines the *Sony S-Log3* log encoding curve / opto-electronic transfer function.
log_decoding_SLog3(y[, bit_depth, ...])	Defines the *Sony S-Log3* log decoding curve / electro-optical transfer function.
log_encoding_VLog(L_in[, bit_depth, ...])	Defines the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.
log_decoding_VLog(V_out[, bit_depth, ...])	Defines the *Panasonic V-Log* log decoding curve / electro-optical transfer function.
log_encoding_ViperLog(x)	Defines the *Viper Log* log encoding curve / opto-electronic transfer function.
log_decoding_ViperLog(y)	Defines the *Viper Log* log decoding curve / electro-optical transfer function.

colour.models.log_encoding_ACEScc

colour.models.**log_encoding_ACEScc**(*lin_AP1*)

Defines the *ACEScc* colourspace log encoding / opto-electronic transfer function.

Parameters **lin_AP1** (numeric or array_like) – *lin_AP1* value.

Returns *ACEScc* non-linear value.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ACEScc	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_encoding_ACEScc(0.18)
0.4135884...
```

colour.models.log_decoding_ACEScc

colour.models.**log_decoding_ACEScc**(*ACEScc*)

Defines the *ACEScc* colourspace log decoding / electro-optical transfer function.

Parameters **ACEScc** (numeric or array_like) – *ACEScc* non-linear value.

Returns *lin_AP1* value.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
ACEScc	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_decoding_ACEScc(0.413588402492442)
0.1799999...
```

colour.models.log_encoding_ACEScct

colour.models.**log_encoding_ACEScct**(*lin_AP1*, *constants={'A': 10.5402377416545, 'B': 0.0729055341958355, 'X_BRK': 0.0078125, 'Y_BRK': 0.155251141552511}*)

Defines the *ACEScct* colourspace log encoding / opto-electronic transfer function.

Parameters

- **lin_AP1** (numeric or array_like) – *lin_AP1* value.
- **constants** (Structure, optional) – *ACEScct* constants.

Returns *ACEScct* non-linear value.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ACEScct	[0, 1]	[0, 1]

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_encoding_ACEScct(0.18)
0.4135884...
```

colour.models.log_decoding_ACEScct

colour.models.**log_decoding_ACEScct**(*ACEScct*, *constants={'A': 10.5402377416545, 'B': 0.0729055341958355, 'X_BRK': 0.0078125, 'Y_BRK': 0.155251141552511}*)

Defines the *ACEScct* colourspace log decoding / electro-optical transfer function.

Parameters

- **ACEScct** (numeric or array_like) – *ACEScct* non-linear value.
- **constants** (Structure, optional) – *ACEScct* constants.

Returns *lin_AP1* value.

Return type numeric or ndarray

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Notes

Domain	Scale - Reference	Scale - 1
ACEScct	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Examples

```
>>> log_decoding_ACEScct(0.413588402492442)
0.1799999...
```

colour.models.log_encoding_ACESproxy

colour.models.**log_encoding_ACESproxy**(*lin_AP1*, *bit_depth=10*, *out_int=False*, *constants={10: {'CV_max': 940, 'CV_min': 64, 'mid_CV_offset': 425, 'mid_log_offset': 2.5, 'steps_per_stop': 50}, 12: {'CV_max': 3760, 'CV_min': 256, 'mid_CV_offset': 1700, 'mid_log_offset': 2.5, 'steps_per_stop': 200}}*)

Defines the *ACESproxy* colourspace log encoding curve / opto-electronic transfer function.

Parameters

- **lin_AP1** (numeric or array_like) – *lin_AP1* value.
- **bit_depth** (int, optional) – **{10, 12}**, *ACESproxy* bit depth.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.
- **constants** (Structure, optional) – *ACESproxy* constants.

Returns *ACESproxy* non-linear value.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
ACESproxy	[0, 1]	[0, 1]

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_encoding_ACESproxy(0.18)
0.4164222...
>>> log_encoding_ACESproxy(0.18, out_int=True)
426
```

colour.models.log_decoding_ACESproxy

colour.models.**log_decoding_ACESproxy**(*ACESproxy*, *bit_depth=10*, *in_int=False*, *constants={10: {'CV_max': 940, 'CV_min': 64, 'mid_CV_offset': 425, 'mid_log_offset': 2.5, 'steps_per_stop': 50}, 12: {'CV_max': 3760, 'CV_min': 256, 'mid_CV_offset': 1700, 'mid_log_offset': 2.5, 'steps_per_stop': 200}}*)

Defines the *ACESproxy* colourspace log decoding curve / electro-optical transfer function.

Parameters

- **ACESproxy** (numeric or array_like) – *ACESproxy* non-linear value.
- **bit_depth** (int, optional) – **{10, 12}**, *ACESproxy* bit depth.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **constants** (Structure, optional) – *ACESproxy* constants.

Returns *lin_AP1* value.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
ACESproxy	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
lin_AP1	[0, 1]	[0, 1]

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a],
[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> log_decoding_ACESproxy(0.416422287390029)
0.1...
>>> log_decoding_ACESproxy(426, in_int=True)
0.1...
```

colour.models.log_encoding_ALEXALogC

colour.models.**log_encoding_ALEXALogC**(*x*, *firmware='SUP 3.x'*, *method='Linear Scene Exposure Factor'*, *EI=800*)

Defines the *ARRI ALEXA Log C* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **firmware** (unicode, optional) – **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **method** (unicode, optional) – **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.
- **EI** (int, optional) – Ei.

Returns *ARRI ALEXA Log C* encoded data t.

Return type numeric or ndarray

References

[ARR12]

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
t	[0, 1]	[0, 1]

Examples

```
>>> log_encoding_ALEXALogC(0.18)
0.3910068...
```

colour.models.log_decoding_ALEXALogC

colour.models.**log_decoding_ALEXALogC**(*t*, *firmware='SUP 3.x'*, *method='Linear Scene Exposure Factor'*, *EI=800*)

Defines the *ARRI ALEXA Log C* log decoding curve / electro-optical transfer function.

Parameters

- **t** (numeric or array_like) – *ARRI ALEXA Log C* encoded data t.
- **firmware** (unicode, optional) – **{'SUP 3.x', 'SUP 2.x'}**, Alexa firmware version.
- **method** (unicode, optional) – **{'Linear Scene Exposure Factor', 'Normalised Sensor Signal'}**, Conversion method.
- **EI** (int, optional) – Ei.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
t	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[ARR12]

Examples

```
>>> log_decoding_ALEXALogC(0.391006832034084)
0.18...
```

colour.models.log_encoding_CanonLog2

colour.models.**log_encoding_CanonLog2**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

Defines the *Canon Log 2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded as normalised code values.

- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *Canon Log 2* non-linear data.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_encoding_CanonLog2(0.18) * 100
39.8254694...
```

colour.models.log_decoding_CanonLog2

colour.models.**log_decoding_CanonLog2**(*clog2*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

Defines the *Canon Log 2* log decoding curve / electro-optical transfer function.

Parameters

- **clog2** (numeric or array_like) – *Canon Log 2* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the *Canon Log 2* non-linear data is encoded with normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog2	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_decoding_CanonLog2(39.825469498316735 / 100)
0.1799999...
```

colour.models.log_encoding_CanonLog3

colour.models.**log_encoding_CanonLog3**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

Defines the *Canon Log 3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *Canon Log 3* non-linear data.

Return type numeric or ndarray

Notes

- Introspection of the grafting points by Shaw, N. (2018) shows that the *Canon Log 3* IDT was likely derived from its encoding curve as the later is grafted at *+/-0.014*:

```
>>> clog3 = 0.04076162
>>> (clog3 - 0.073059361) / 2.3069815
-0.014000000000000002
>>> clog3 = 0.105357102
>>> (clog3 - 0.073059361) / 2.3069815
0.013999999999999997
```

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog3	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_encoding_CanonLog3(0.18) * 100
34.3389369...
```

colour.models.log_decoding_CanonLog3

colour.models.**log_decoding_CanonLog3**(*clog3*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

Defines the *Canon Log 3* log decoding curve / electro-optical transfer function.

Parameters

- **clog3** (numeric or array_like) – *Canon Log 3* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the *Canon Log 3* non-linear data is encoded with normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog3	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Can16]

Examples

```
>>> log_decoding_CanonLog3(34.338936938868677 / 100)
0.1800000...
```

colour.models.log_encoding_CanonLog

colour.models.**log_encoding_CanonLog**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

Defines the *Canon Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the *Canon Log* non-linear data is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns *Canon Log* non-linear data.

Return type numeric or ndarray

References

[Tho12]

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
clog	[0, 1]	[0, 1]

Examples

```
>>> log_encoding_CanonLog(0.18) * 100
34.3389651...
```

The values of *Table 2 Canon-Log Code Values* table in [Tho12] are obtained as follows:

```
>>> x = np.array([0, 2, 18, 90, 720]) / 100
>>> np.around(log_encoding_CanonLog(x) * (2 ** 10 - 1)).astype(np.int)
array([ 128,  169,  351,  614, 1016])
>>> np.around(log_encoding_CanonLog(x, 10, False) * 100, 1)
array([  7.3,  12. ,  32.8,  62.7,  108.7])
```

colour.models.log_decoding_CanonLog

colour.models.**log_decoding_CanonLog**(*clog*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

Defines the *Canon Log* log decoding curve / electro-optical transfer function.

Parameters

- **clog** (numeric or array_like) – *Canon Log* non-linear data.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the *Canon Log* non-linear data is encoded with normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
clog	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Tho12]

Examples

```
>>> log_decoding_CanonLog(34.338965172606912 / 100)
0.17999999...
```

colour.models.log_encoding_Cineon

colour.models.**log_encoding_Cineon**(*x*, *black_offset=0.0107977516232771*)

Defines the *Cineon* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_Cineon(0.18)
0.4573196...
```

colour.models.log_decoding_Cineon

colour.models.**log_decoding_Cineon**(*y*, *black_offset=0.0107977516232771*)

Defines the *Cineon* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_Cineon(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_ERIMMRGB

colour.models.**log_encoding_ERIMMRGB**(*X*, *bit_depth=8*, *out_int=False*, *E_min=0.001*, *E_clip=316.2*)

Defines the *ERIMM RGB* log encoding curve / opto-electronic transfer function (OETF / OECF).

Parameters

- **X** (numeric or array_like) – Linear data X_{ERIMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.
- **E_min** (numeric, optional) – Minimum exposure limit.
- **E_clip** (numeric, optional) – Maximum exposure limit.

Returns Non-linear data X'_{ERIMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

* This definition has an output integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> log_encoding_ERIMMRGB(0.18)
0.4100523...
>>> log_encoding_ERIMMRGB(0.18, out_int=True)
105
```

colour.models.log_decoding_ERIMMRGB

colour.models.**log_decoding_ERIMMRGB**(*X_p*, *bit_depth=8*, *in_int=False*, *E_min=0.001*, *E_clip=316.2*)

Defines the *ERIMM RGB* log decoding curve / electro-optical transfer function (EOTF / EOCF).

Parameters

- **X_p** (numeric or array_like) – Non-linear data X'_{ERIMM}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **E_min** (numeric, optional) – Minimum exposure limit.
- **E_clip** (numeric, optional) – Maximum exposure limit.

Returns Linear data X_{ERIMM}.

Return type numeric or ndarray

Notes

Domain *	Scale - Reference	Scale - 1
X_p	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
X	[0, 1]	[0, 1]

* This definition has an input integer switch, thus the domain-range scale information is only given for the floating point mode.

References

[SWG00]

Examples

```
>>> log_decoding_ERIMMRGB(0.410052389492129)
0.1...
>>> log_decoding_ERIMMRGB(105, in_int=True)
0.1...
```

colour.models.log_encoding_FLog

colour.models.**log_encoding_FLog**(*in_r*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, *constants={'a': 0.555556, 'b': 0.009468, 'c': 0.344676, 'cut1': 0.00089, 'cut2': 0.100537775223865, 'd': 0.790453, 'e': 8.735631, 'f': 0.092864}*)

Defines the *Fujifilm F-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **in_r** (numeric or array_like) – Linear reflection data :math`in`.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Fujifilm F-Log* data out is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level :math`in` to a camera is reflection.
- **constants** (Structure, optional) – *Fujifilm F-Log* constants.

Returns Non-linear data out.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

References

[Fuj16]

Examples

```
>>> log_encoding_FLog(0.18)
0.4593184...
```

The values of *2-2. F-Log Code Value* table in [Fuj16] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_FLog(x, 10, False) * 100, 1)
array([  3.5,  46.3,  73.2])
>>> np.around(log_encoding_FLog(x) * (2 ** 10 - 1)).astype(np.int)
array([ 95, 470, 705])
```

colour.models.log_decoding_FLog

colour.models.**log_decoding_FLog**(*out_r*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, *constants={'a': 0.555556, 'b': 0.009468, 'c': 0.344676, 'cut1': 0.00089, 'cut2': 0.100537775223865, 'd': 0.790453, 'e': 8.735631, 'f': 0.092864}*)

Defines the *Fujifilm F-Log* log decoding curve / electro-optical transfer function.

Parameters

- **out_r** (numeric or array_like) – Non-linear data out.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the non-linear *Fujifilm F-Log* data out is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level :math`in` to a camera is reflection.
- **constants** (Structure, optional) – *Fujifilm F-Log* constants.

Returns Linear reflection data :math`in`.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
out_r	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
in_r	[0, 1]	[0, 1]

References

[Fuj16]

Examples

```
>>> log_decoding_FLog(0.45931845866162124)
0.1800000...
```

colour.models.LOG3G10_ENCODING_METHODS

colour.models.**LOG3G10_ENCODING_METHODS = CaseInsensitiveMapping({'v1': ..., 'v2': ...})**

Supported *Log3G10* log encoding curve / opto-electronic transfer function methods.

References

[Nat16]

LOG3G10_ENCODING_METHODS [CaseInsensitiveMapping] **{'v1', 'v2'}**

colour.models.log_encoding_Log3G10

colour.models.**log_encoding_Log3G10**(*x*, *method='v2'*, ***kwargs*)

Defines the *Log3G10* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **method** (unicode, optional) – **{'v1', 'v2'}**, Computation method.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

- The *Log3G10 v1* log encoding curve is the one used in *REDCINE-X Beta 42*. *Resolve 12.5.2* also uses the *v1* curve. *RED* is planning to use the *Log3G10 v2* log encoding curve in the release version of the *RED SDK*.
- The intent of the *Log3G10 v1* log encoding curve is that zero maps to zero, 0.18 maps to 1/3, and 10 stops above 0.18 maps to 1.0. The name indicates this in a similar way to the naming conventions of *Sony HyperGamma* curves.

  The constants used in the functions do not in fact quite hit these values, but rather than use corrected constants, the functions here use the official *RED* values, in order to match the output of the *RED SDK*.

  For those interested, solving for constants which exactly hit 1/3 and 1.0 yields the following values:

```
B = 25 * (np.sqrt(4093.0) - 3) / 9
A = 1 / np.log10(B * 184.32 + 1)
```

  where the function takes the form:

```
Log3G10(x) = A * np.log10(B * x + 1)
```

  Similarly for *Log3G12*, the values which hit exactly 1/3 and 1.0 are:

```
B = 25 * (np.sqrt(16381.0) - 3) / 9
A = 1 / np.log10(B * 737.28 + 1)
```

References

[Nat16]

Examples

```
>>> log_encoding_Log3G10(0.0)
0.0915514...
>>> log_encoding_Log3G10(0.18, method='v1')
0.3333336...
```

colour.models.LOG3G10_DECODING_METHODS

colour.models.**LOG3G10_DECODING_METHODS = CaseInsensitiveMapping({'v1': ..., 'v2': ...})**
Supported *Log3G10* log decoding curve / electro-optical transfer function methods.

References

[Nat16]

LOG3G10_DECODING_METHODS [CaseInsensitiveMapping] **{'v1', 'v2'}**

colour.models.log_decoding_Log3G10

colour.models.**log_decoding_Log3G10**(*y*, *method='v2'*, ***kwargs*)
Defines the *Log3G10* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **method** (unicode, optional) – **{'v1', 'v2'}**, Computation method.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nat16]

Examples

```
>>> log_decoding_Log3G10(1.0)
184.3223476...
>>> log_decoding_Log3G10(1.0 / 3, method='v1')
0.1799994...
```

colour.models.log_encoding_Log3G12

colour.models.**log_encoding_Log3G12**(*x*)

Defines the *Log3G12* log encoding curve / opto-electronic transfer function.

Parameters **x** (numeric or array_like) – Linear data x.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[Nat16]

Examples

```
>>> log_encoding_Log3G12(0.18)
0.3333326...
```

colour.models.log_decoding_Log3G12

colour.models.**log_decoding_Log3G12**(*y*)

Defines the *Log3G12* log decoding curve / electro-optical transfer function.

Parameters **y** (numeric or array_like) – Non-linear data y.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[Nat16]

Examples

```
>>> log_decoding_Log3G12(1.0 / 3)
0.1800015...
```

colour.models.log_encoding_Panalog

colour.models.**log_encoding_Panalog**(*x*, *black_offset=0.04077184461038074*)

Defines the *Panalog* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data y.

Return type numeric or ndarray

Warning: These are estimations known to be close enough, the actual log encoding curves are not published.

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_Panalog(0.18)
0.3745767...
```

colour.models.log_decoding_Panalog

colour.models.**log_decoding_Panalog**(*y*, *black_offset=0.04077184461038074*)

Defines the *Panalog* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x.

Return type numeric or ndarray

Warning: These are estimations known to be close enough, the actual log encoding curves are not published.

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_Panalog(0.374576791382298)
0.1...
```

colour.models.log_encoding_PivotedLog

colour.models.**log_encoding_PivotedLog**(*x*, *log_reference=445*, *linear_reference=0.18*, *negative_gamma=0.6*, *density_per_code_value=0.002*)

Defines the *Josh Pines* style *Pivoted Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **log_reference** (numeric or array_like) – Log reference.
- **linear_reference** (numeric or array_like) – Linear reference.
- **negative_gamma** (numeric or array_like) – Negative gamma.
- **density_per_code_value** (numeric or array_like) – Density per code value.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_PivotedLog(0.18)
0.4349951...
```

colour.models.log_decoding_PivotedLog

colour.models.**log_decoding_PivotedLog**(*y*, *log_reference=445*, *linear_reference=0.18*, *negative_gamma=0.6*, *density_per_code_value=0.002*)

Defines the *Josh Pines* style *Pivoted Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **log_reference** (numeric or array_like) – Log reference.
- **linear_reference** (numeric or array_like) – Linear reference.
- **negative_gamma** (numeric or array_like) – Negative gamma.
- **density_per_code_value** (numeric or array_like) – Density per code value.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_PivotedLog(0.434995112414467)
0.1...
```

colour.models.log_encoding_Protune

colour.models.**log_encoding_Protune**(x)

Defines the *Protune* log encoding curve / opto-electronic transfer function.

Parameters **x** (numeric or array_like) – Linear data x.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[GDM16]

Examples

```
>>> log_encoding_Protune(0.18)
0.6456234...
```

colour.models.log_decoding_Protune

colour.models.**log_decoding_Protune**(y)

Defines the *Protune* log decoding curve / electro-optical transfer function.

Parameters **y** (numeric or array_like) – Non-linear data y.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[GDM16]

Examples

```
>>> log_decoding_Protune(0.645623486803636)
0.1...
```

colour.models.log_encoding_REDLog

colour.models.**log_encoding_REDLog**(x, *black_offset=0.009955040995908344*)

Defines the *REDLog* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_REDLog(0.18)
0.6376218...
```

colour.models.log_decoding_REDLog

colour.models.**log_decoding_REDLog**(*y*, *black_offset=0.009955040995908344*)
Defines the *REDLog* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_REDLog(0.637621845988175)
0.1...
```

colour.models.log_encoding_REDLogFilm

colour.models.**log_encoding_REDLogFilm**(*x*, *black_offset=0.0107977516232771*)
Defines the *REDLogFilm* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Linear data x.
- **black_offset** (numeric or array_like) – Black offset.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_REDLogFilm(0.18)
0.4573196...
```

colour.models.log_decoding_REDLogFilm

colour.models.**log_decoding_REDLogFilm**(*y*, *black_offset=0.0107977516232771*)

Defines the *REDLogFilm* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear data y.
- **black_offset** (numeric or array_like) – Black offset.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_REDLogFilm(0.457319613085418)
0.1799999...
```

colour.models.log_encoding_SLog

colour.models.**log_encoding_SLog**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

Defines the *Sony S-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Non-linear *Sony S-Log* data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_encoding_SLog(0.18)
0.3849708...
```

The values of *IRE and CV of S-Log2 @ISO800* table in [SonyCorporation12] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog(x, 10, False) * 100).astype(np.int)
array([ 3, 38, 65])
>>> np.around(log_encoding_SLog(x) * (2 ** 10 - 1)).astype(np.int)
array([ 90, 394, 636])
```

colour.models.log_decoding_SLog

colour.models.**log_decoding_SLog**(*y*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

Defines the *Sony S-Log* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log* data y.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log* data y is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_decoding_SLog(0.384970815928670)
0.1...
```

colour.models.log_encoding_SLog2

colour.models.**log_encoding_SLog2**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

Defines the *Sony S-Log2* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log2* data y is encoded as normalised code values.

- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Non-linear *Sony S-Log2* data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_encoding_SLog2(0.18)
0.3395325...
```

The values of *IRE and CV of S-Log2 @ISO800* table in [SonyCorporation12] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog2(x, 10, False) * 100).astype(np.int)
array([ 3, 32, 59])
>>> np.around(log_encoding_SLog2(x) * (2 ** 10 - 1)).astype(np.int)
array([ 90, 347, 582])
```

colour.models.log_decoding_SLog2

colour.models.**log_decoding_SLog2**(*y*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

Defines the *Sony S-Log2* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log2* data y.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log2* data y is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporation12]

Examples

```
>>> log_decoding_SLog2(0.339532524633774)
0.1...
```

colour.models.log_encoding_SLog3

colour.models.**log_encoding_SLog3**(*x*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, ***kwargs*)

Defines the *Sony S-Log3* log encoding curve / opto-electronic transfer function.

Parameters

- **x** (numeric or array_like) – Reflection or $IRE/100$ input light level x to a camera.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log3* data y is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Non-linear *Sony S-Log3* data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyCorporationc]

Examples

```
>>> log_encoding_SLog3(0.18)
0.4105571...
```

The values of *S-Log3 10bit code values (18%, 90%)* table in [SonyCorporationc] are obtained as follows:

```
>>> x = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_SLog3(x, 10, False) * 100).astype(np.int)
array([ 4, 41, 61])
>>> np.around(log_encoding_SLog3(x) * (2 ** 10 - 1)).astype(np.int)
array([ 95, 420, 598])
```

colour.models.log_decoding_SLog3

colour.models.**log_decoding_SLog3**(*y*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, ***kwargs*)

Defines the *Sony S-Log3* log decoding curve / electro-optical transfer function.

Parameters

- **y** (numeric or array_like) – Non-linear *Sony S-Log3* data y.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the non-linear *Sony S-Log3* data y is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level x to a camera is reflection.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Reflection or $IRE/100$ input light level x to a camera.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyCorporationc]

Examples

```
>>> log_decoding_SLog3(0.410557184750733)
0.1...
```

colour.models.log_encoding_VLog

colour.models.**log_encoding_VLog**(*L_in*, *bit_depth=10*, *out_normalised_code_value=True*, *in_reflection=True*, *constants={'b': 0.00873, 'c': 0.241514, 'cut1': 0.01, 'cut2': 0.181, 'd': 0.598206}*, ***kwargs*)

Defines the *Panasonic V-Log* log encoding curve / opto-electronic transfer function.

Parameters

- **L_in** (numeric or array_like) – Linear reflection data :math`L_{in}`.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **out_normalised_code_value** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded as normalised code values.
- **in_reflection** (bool, optional) – Whether the light level :math`L_{in}` to a camera is reflection.
- **constants** (Structure, optional) – *Panasonic V-Log* constants.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Non-linear data V_{out}.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
L_in	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
V_out	[0, 1]	[0, 1]

References

[Pan14]

Examples

```
>>> log_encoding_VLog(0.18)
0.4233114...
```

The values of *Fig.2.2 V-Log Code Value* table in [Pan14] are obtained as follows:

```
>>> L_in = np.array([0, 18, 90]) / 100
>>> np.around(log_encoding_VLog(L_in, 10, False) * 100).astype(np.int)
array([ 7, 42, 61])
>>> np.around(log_encoding_VLog(L_in) * (2 ** 10 - 1)).astype(np.int)
array([128, 433, 602])
>>> np.around(log_encoding_VLog(L_in) * (2 ** 12 - 1)).astype(np.int)
array([ 512, 1733, 2409])
```

Note that some values in the last column values of *Fig.2.2 V-Log Code Value* table in [Pan14] are different by a code: [512, 1732, 2408].

colour.models.log_decoding_VLog

colour.models.**log_decoding_VLog**(*V_out*, *bit_depth=10*, *in_normalised_code_value=True*, *out_reflection=True*, *constants={'b': 0.00873, 'c': 0.241514, 'cut1': 0.01, 'cut2': 0.181, 'd': 0.598206}*, ***kwargs*)

Defines the *Panasonic V-Log* log decoding curve / electro-optical transfer function.

Parameters

- **V_out** (numeric or array_like) – Non-linear data V_{out}.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_normalised_code_value** (bool, optional) – Whether the non-linear *Panasonic V-Log* data V_{out} is encoded as normalised code values.
- **out_reflection** (bool, optional) – Whether the light level :math`L_{in}` to a camera is reflection.
- **constants** (Structure, optional) – *Panasonic V-Log* constants.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments for deprecation management.

Returns Linear reflection data :math`L_{in}`.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
V_out	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
L_in	[0, 1]	[0, 1]

References

[Pan14]

Examples

```
>>> log_decoding_VLog(0.423311448760136)
0.1799999...
```

colour.models.log_encoding_ViperLog

colour.models.**log_encoding_ViperLog**(x)

Defines the *Viper Log* log encoding curve / opto-electronic transfer function.

Parameters **x** (numeric or array_like) – Linear data x.

Returns Non-linear data y.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_encoding_ViperLog(0.18)
0.6360080...
```

colour.models.log_decoding_ViperLog

colour.models.**log_decoding_ViperLog**(y)

Defines the *Viper Log* log decoding curve / electro-optical transfer function.

Parameters **y** (numeric or array_like) – Non-linear data y.

Returns Linear data x.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
y	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
x	[0, 1]	[0, 1]

References

[SonyImageworks12]

Examples

```
>>> log_decoding_ViperLog(0.636008067010413)
0.1799999...
```

ACES Spectral Conversion

colour

sd_to_aces_relative_exposure_values(sd[, ...])	Converts given spectral distribution to *ACES2065-1* colourspace relative exposure values.

colour.sd_to_aces_relative_exposure_values

colour.**sd_to_aces_relative_exposure_values**(*sd*, *illuminant=SpectralDistribution(name='D65', ...)*, *apply_chromatic_adaptation=False*, *chromatic_adaptation_transform='CAT02'*)

Converts given spectral distribution to *ACES2065-1* colourspace relative exposure values.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution.
- **illuminant** (SpectralDistribution, optional) – *Illuminant* spectral distribution.
- **apply_chromatic_adaptation** (bool, optional) – Whether to apply chromatic adaptation using given transform.
- **chromatic_adaptation_transform** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* transform.

Returns *ACES2065-1* colourspace relative exposure values array.

Return type ndarray, (3,)

Notes

Range	Scale - Reference	Scale - 1
XYZ	[0, 100]	[0, 1]

- The chromatic adaptation method implemented here is a bit unusual as it involves building a new colourspace based on *ACES2065-1* colourspace primaries but using the whitepoint of the illuminant that the spectral distribution was measured under.

References

[For18], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d], [TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee]

Examples

```
>>> from colour import COLOURCHECKERS_SDS
>>> sd = COLOURCHECKERS_SDS['ColorChecker N Ohta']['dark skin']
>>> sd_to_aces_relative_exposure_values(sd)
array([ 0.1171785...,  0.0866347...,  0.0589707...])
>>> sd_to_aces_relative_exposure_values(sd,
...     apply_chromatic_adaptation=True)
array([ 0.1180766...,  0.0869023...,  0.0589104...])
```

Ancillary Objects

`colour.models`

ACES_RICD	Implements support for the *CIE RGB* colour matching functions.

colour.models.ACES_RICD

`colour.models.`**`ACES_RICD = RGB_ColourMatchingFunctions(name='ACES RICD', ...)`**

Implements support for the *CIE RGB* colour matching functions.

Parameters

- **data** (Series or Dataframe or Signal or MultiSignals or MultiSpectralDistributions or array_like or dict_like, optional) – Data to be stored in the multi-spectral distributions.
- **domain** (array_like, optional) – Values to initialise the multiple colour.SpectralDistribution class instances colour.continuous.Signal.wavelengths attribute with. If both data and domain arguments are defined, the latter will be used to initialise the colour.continuous.Signal.wavelengths attribute.
- **labels** (array_like, optional) – Names to use for the colour.SpectralDistribution class instances.

Other Parameters

- **name** (*unicode, optional*) – Multi-spectral distributions name.
- **interpolator** (*object, optional*) – Interpolator class type to use as interpolating function for the colour.SpectralDistribution class instances.

- **interpolator_args** (*dict_like, optional*) – Arguments to use when instantiating the interpolating function of the `colour.SpectralDistribution` class instances.
- **extrapolator** (*object, optional*) – Extrapolator class type to use as extrapolating function for the `colour.SpectralDistribution` class instances.
- **extrapolator_args** (*dict_like, optional*) – Arguments to use when instantiating the extrapolating function of the `colour.SpectralDistribution` class instances.
- **strict_labels** (*array_like, optional*) – Multi-spectral distributions labels for figures, default to `colour.colorimetry.RGB_ColourMatchingFunctions.labels` attribute value.

Colour Encodings

Y'CbCr Colour Encoding

`colour`

`RGB_to_YCbCr`(RGB[, K, in_bits, in_legal, ...])	Converts an array of *R'G'B'* values to the corresponding *Y'CbCr* colour encoding values array.
`YCbCr_to_RGB`(YCbCr[, K, in_bits, in_legal, ...])	Converts an array of *Y'CbCr* colour encoding values to the corresponding *R'G'B'* values array.
`YCBCR_WEIGHTS`	Implements a case-insensitive mutable mapping / *dict* object.
`RGB_to_YcCbcCrc`(RGB[, out_bits, out_legal, ...])	Converts an array of *RGB* linear values to the corresponding *Yc'Cbc'Crc'* colour encoding values array.
`YcCbcCrc_to_RGB`(YcCbcCrc[, in_bits, ...])	Converts an array of *Yc'Cbc'Crc'* colour encoding values to the corresponding *RGB* array of linear values.

colour.RGB_to_YCbCr

`colour.`**`RGB_to_YCbCr`**(*RGB*, *K=array([0.2126, 0.0722])*, *in_bits=10*, *in_legal=False*, *in_int=False*, *out_bits=8*, *out_legal=True*, *out_int=False*, ***kwargs*)

Converts an array of *R'G'B'* values to the corresponding *Y'CbCr* colour encoding values array.

Parameters

- **RGB** (`array_like`) – Input *R'G'B'* array of floats or integer values.
- **K** (`array_like, optional`) – Luma weighting coefficients of red and blue. See `colour.YCBCR_WEIGHTS` for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (`int, optional`) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Default is *10*.
- **in_legal** (`bool, optional`) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (`bool, optional`) – Whether to treat the input values as `in_bits` integer code values. Default is *False*.
- **out_bits** (`int, optional`) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Ignored if `out_legal` and `out_int` are both *False*. Default is *8*.

- **out_legal** (bool, optional) – Whether to return legal range values. Default is *True*.
- **out_int** (bool, optional) – Whether to return values as out_bits integer code values. Default is *False*.

Other Parameters

- **in_range** (*array_like, optional*) – Array overriding the computed range such as *in_range = (RGB_min, RGB_max)*. If in_range is undefined, *RGB_min* and *RGB_max* will be computed using colour.CV_range() definition.
- **out_range** (*array_like, optional*) – Array overriding the computed range such as *out_range = (Y_min, Y_max, C_min, C_max)*`. *If ``out_range`` is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using colour.models.rgb.ycbcr.YCbCr_ranges() definition.

Returns *Y'CbCr* colour encoding array of integer or float values.

Return type ndarray

Warning: For *Recommendation ITU-R BT.2020*, colour.RGB_to_YCbCr() definition is only applicable to the non-constant luminance implementation. colour.RGB_to_YcCbcCrc() definition should be used for the constant luminance case as per [InternationalTUnion15a].

Notes

Domain *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
YCbCr	[0, 1]	[0, 1]

* This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

- The default arguments, **{'in_bits': 10, 'in_legal': False, 'in_int': False, 'out_bits': 8, 'out_legal': True, 'out_int': False} transform a float *R'G'B'* input array normalised to domain [0, 1] (in_bits is ignored) to a float *Y'CbCr* output array where *Y'* is normalised to range [16 / 255, 235 / 255] and *Cb* and *Cr* are normalised to range [16 / 255, 240./255]. The float values are calculated based on an [0, 255] integer range, but no 8-bit quantisation or clamping are performed.

References

[InternationalTUnion11c], [InternationalTUnion15b], [SocietyoMPaTEngineers99], [Wik04e]

Examples

```
>>> RGB = np.array([1.0, 1.0, 1.0])
>>> RGB_to_YCbCr(RGB)
array([ 0.9215686...,  0.5019607...,  0.5019607...])
```

Matching float output of The Foundry Nuke's Colorspace node set to YCbCr:

```
>>> RGB_to_YCbCr(RGB,
...              out_range=(16 / 255, 235 / 255, 15.5 / 255, 239.5 / 255))
...
array([ 0.9215686...,  0.5       ,  0.5       ])
```

Matching float output of The Foundry Nuke's Colorspace node set to YPbPr:

```
>>> RGB_to_YCbCr(RGB, out_legal=False, out_int=False)
...
array([ 1.,  0.,  0.])
```

Creating integer code values as per standard 10-bit SDI:

```
>>> RGB_to_YCbCr(RGB, out_legal=True, out_bits=10, out_int=True)
array([940, 512, 512])
```

For JFIF JPEG conversion as per ITU-T T.871 [InternationalTUnion11c]:

```
>>> RGB = np.array([102, 0, 51])
>>> RGB_to_YCbCr(RGB, K=YCBCR_WEIGHTS['ITU-R BT.601'], in_range=(0, 255),
...              out_range=(0, 255, 0, 256), out_int=True)
array([ 36, 136, 175])
```

Note the use of 256 for the max *Cb / Cr* value, which is required so that the *Cb* and *Cr* output is centered about 128. Using 255 centres it about 127.5, meaning that there is no integer code value to represent achromatic colours. This does however create the possibility of output integer codes with value of 256, which cannot be stored in 8-bit integer representation. Recommendation ITU-T T.871 specifies these should be clamped to 255.

These JFIF JPEG ranges are also obtained as follows:

```
>>> RGB_to_YCbCr(RGB, K=YCBCR_WEIGHTS['ITU-R BT.601'], in_bits=8,
...              in_int=True, out_legal=False, out_int=True)
array([ 36, 136, 175])
```

colour.YCbCr_to_RGB

colour.**YCbCr_to_RGB**(*YCbCr*, *K=array([0.2126, 0.0722])*, *in_bits=8*, *in_legal=True*, *in_int=False*, *out_bits=10*, *out_legal=False*, *out_int=False*, ***kwargs*)

Converts an array of *Y'CbCr* colour encoding values to the corresponding *R'G'B'* values array.

Parameters

- **YCbCr** (array_like) – Input *Y'CbCr* colour encoding array of integer or float values.
- **K** (array_like, optional) – Luma weighting coefficients of red and blue. See colour.YCBCR_WEIGHTS for presets. Default is *(0.2126, 0.0722)*, the weightings for *ITU-R BT.709*.
- **in_bits** (int, optional) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Default is *8*.

- **in_legal** (bool, optional) – Whether to treat the input values as legal range. Default is *True*.
- **in_int** (bool, optional) – Whether to treat the input values as in_bits integer code values. Default is *False*.
- **out_bits** (int, optional) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Ignored if out_legal and out_int are both *False*. Default is *10*.
- **out_legal** (bool, optional) – Whether to return legal range values. Default is *False*.
- **out_int** (bool, optional) – Whether to return values as out_bits integer code values. Default is *False*.

Other Parameters

- **in_range** (*array_like, optional*) – Array overriding the computed range such as *in_range = (Y_min, Y_max, C_min, C_max)*. If in_range is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using colour.models.rgb.ycbcr.YCbCr_ranges() definition.
- **out_range** (*array_like, optional*) – Array overriding the computed range such as *out_range = (RGB_min, RGB_max)*. If out_range is undefined, *RGB_min* and *RGB_max* will be computed using colour.CV_range() definition.

Returns *R'G'B'* array of integer or float values.

Return type ndarray

Notes

Domain *	Scale - Reference	Scale - 1
YCbCr	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

* This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

Warning: For *Recommendation ITU-R BT.2020*, colour.YCbCr_to_RGB() definition is only applicable to the non-constant luminance implementation. colour.YcCbcCrc_to_RGB() definition should be used for the constant luminance case as per [InternationalTUnion15a].

References

[InternationalTUnion11c], [InternationalTUnion15b], [SocietyoMPaTEngineers99], [Wik04e]

Examples

```
>>> YCbCr = np.array([502, 512, 512])
>>> YCbCr_to_RGB(YCbCr, in_bits=10, in_legal=True, in_int=True)
array([ 0.5,  0.5,  0.5])
```

colour.YCBCR_WEIGHTS

colour.**YCBCR_WEIGHTS = CaseInsensitiveMapping({'ITU-R BT.601': ..., 'ITU-R BT.709': ..., 'ITU-R BT.2020'**

Implements a case-insensitive mutable mapping / *dict* object.

Allows values retrieving from keys while ignoring the key case. The keys are expected to be unicode or string-like objects supporting the `str.lower()` method.

Parameters **data** (`dict`) – *dict* of data to store into the mapping at initialisation.

Other Parameters ****kwargs** (*dict, optional*) – Key / Value pairs to store into the mapping at initialisation.

colour.**__setitem__**()

colour.**__getitem__**()

colour.**__delitem__**()

colour.**__contains__**()

colour.**__iter__**()

colour.**__len__**()

colour.**__eq__**()

colour.**__ne__**()

colour.**__repr__**()

colour.**copy**()

colour.**lower_items**()

Warning: The keys are expected to be unicode or string-like objects.

References

[Rei]

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

colour.RGB_to_YcCbcCrc

colour.**RGB_to_YcCbcCrc**(*RGB*, *out_bits=10*, *out_legal=True*, *out_int=False*, *is_12_bits_system=False*, ***kwargs*)

Converts an array of *RGB* linear values to the corresponding *Yc'Cbc'Crc'* colour encoding values array.

Parameters

- **RGB** (array_like) – Input *RGB* array of linear float values.
- **out_bits** (int, optional) – Bit depth for integer output, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Ignored if out_legal and out_int are both *False*. Default is *10*.
- **out_legal** (bool, optional) – Whether to return legal range values. Default is *True*.
- **out_int** (bool, optional) – Whether to return values as out_bits integer code values. Default is *False*.
- **is_12_bits_system** (bool, optional) – *Recommendation ITU-R BT.2020* OETF (OECF) adopts different parameters for 10 and 12 bit systems. Default is *False*.

Other Parameters **out_range** (*array_like, optional*) – Array overriding the computed range such as *out_range = (Y_min, Y_max, C_min, C_max)*. If out_range is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using colour.models.rgb.ycbcr.YCbCr_ranges() definition.

Returns *Yc'Cbc'Crc'* colour encoding array of integer or float values.

Return type ndarray

Notes

Domain *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
YcCbcCrc	[0, 1]	[0, 1]

* This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

[InternationalTUnion15a], [Wik04e]

Examples

```
>>> RGB = np.array([0.18, 0.18, 0.18])
>>> RGB_to_YcCbcCrc(RGB, out_legal=True, out_bits=10, out_int=True,
...                 is_12_bits_system=False)
array([422, 512, 512])
```

colour.YcCbcCrc_to_RGB

colour.**YcCbcCrc_to_RGB**(*YcCbcCrc*, *in_bits=10*, *in_legal=True*, *in_int=False*, *is_12_bits_system=False*, ***kwargs*)

Converts an array of *Yc'Cbc'Crc'* colour encoding values to the corresponding *RGB* array of linear values.

Parameters

- **YcCbcCrc** (array_like) – Input *Yc'Cbc'Crc'* colour encoding array of linear float values.
- **in_bits** (int, optional) – Bit depth for integer input, or used in the calculation of the denominator for legal range float values, i.e. 8-bit means the float value for legal white is *235 / 255*. Default is *10*.
- **in_legal** (bool, optional) – Whether to treat the input values as legal range. Default is *False*.
- **in_int** (bool, optional) – Whether to treat the input values as in_bits integer code values. Default is *False*.
- **is_12_bits_system** (bool, optional) – *Recommendation ITU-R BT.2020* EOTF (EOCF) adopts different parameters for 10 and 12 bit systems. Default is *False*.

Other Parameters **in_range** (*array_like, optional*) – Array overriding the computed range such as *in_range = (Y_min, Y_max, C_min, C_max)*. If in_range is undefined, *Y_min*, *Y_max*, *C_min* and *C_max* will be computed using colour.models.rgb.ycbcr.YCbCr_ranges() definition.

Returns *RGB* array of linear float values.

Return type ndarray

Notes

Domain *	Scale - Reference	Scale - 1
YcCbcCrc	[0, 1]	[0, 1]

Range *	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

* This definition has input and output integer switches, thus the domain-range scale information is only given for the floating point mode.

Warning: This definition is specifically for usage with *Recommendation ITU-R BT.2020* when adopting the constant luminance implementation.

References

[InternationalTUnion15a], [Wik04e]

Examples

```
>>> YcCbcCrc = np.array([1689, 2048, 2048])
>>> YcCbcCrc_to_RGB(YcCbcCrc, in_legal=True, in_bits=12, in_int=True,
...                 is_12_bits_system=True)
...
array([ 0.1800903...,  0.1800903...,  0.1800903...])
```

Ancillary Objects

`colour`

`full_to_legal`(CV[, bit_depth, in_int, out_int])	Converts given code value CV or float equivalent of a code value at a given bit depth from full range (full swing) to legal range (studio swing).
`legal_to_full`(CV[, bit_depth, in_int, out_int])	Converts given code value CV or float equivalent of a code value at a given bit depth from legal range (studio swing) to full range (full swing).
`CV_range`([bit_depth, is_legal, is_int])	Returns the code value CV range for given bit depth, range legality and representation.

colour.full_to_legal

`colour.`**`full_to_legal`**(*CV*, *bit_depth=10*, *in_int=False*, *out_int=False*)

Converts given code value CV or float equivalent of a code value at a given bit depth from full range (full swing) to legal range (studio swing).

Parameters

- **CV** (`array_like`) – Full range code value CV or float equivalent of a code value at a given bit depth.
- **bit_depth** (`int, optional`) – Bit depth used for conversion.
- **in_int** (`bool, optional`) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **out_int** (`bool, optional`) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Legal range code value CV or float equivalent of a code value at a given bit depth.

Return type ndarray

Examples

```
>>> full_to_legal(0.0)
0.0625610...
>>> full_to_legal(1.0)
0.9188660...
>>> full_to_legal(0.0, out_int=True)
64
>>> full_to_legal(1.0, out_int=True)
940
>>> full_to_legal(0, in_int=True)
0.0625610...
>>> full_to_legal(1023, in_int=True)
0.9188660...
>>> full_to_legal(0, in_int=True, out_int=True)
64
>>> full_to_legal(1023, in_int=True, out_int=True)
940
```

colour.legal_to_full

colour.**legal_to_full**(*CV*, *bit_depth=10*, *in_int=False*, *out_int=False*)

Converts given code value CV or float equivalent of a code value at a given bit depth from legal range (studio swing) to full range (full swing).

Parameters

- **CV** (array_like) – Legal range code value CV or float equivalent of a code value at a given bit depth.
- **bit_depth** (int, optional) – Bit depth used for conversion.
- **in_int** (bool, optional) – Whether to treat the input value as integer code value or float equivalent of a code value at a given bit depth.
- **out_int** (bool, optional) – Whether to return value as integer code value or float equivalent of a code value at a given bit depth.

Returns Full range code value CV or float equivalent of a code value at a given bit depth.

Return type ndarray

Examples

```
>>> legal_to_full(64 / 1023)
0.0
>>> legal_to_full(940 / 1023)
1.0
>>> legal_to_full(64 / 1023, out_int=True)
0
>>> legal_to_full(940 / 1023, out_int=True)
1023
>>> legal_to_full(64, in_int=True)
0.0
>>> legal_to_full(940, in_int=True)
1.0
>>> legal_to_full(64, in_int=True, out_int=True)
0
>>> legal_to_full(940, in_int=True, out_int=True)
1023
```

colour.CV_range

colour.**CV_range**(*bit_depth=10*, *is_legal=False*, *is_int=False*)

Returns the code value CV range for given bit depth, range legality and representation.

Parameters

- **bit_depth** (int, optional) – Bit depth of the code value CV range.
- **is_legal** (bool, optional) – Whether the code value CV range is legal.
- **is_int** (bool, optional) – Whether the code value CV range represents integer code values.

Returns Code value CV range.

Return type ndarray

Examples

```
>>> CV_range(8, True, True)
array([ 16, 235])
>>> CV_range(8, True, False)
array([ 0.0627451...,  0.9215686...])
>>> CV_range(10, False, False)
array([ 0.,  1.])
```

YCoCg Colour Encoding

colour

RGB_to_YCoCg(RGB)	Converts an array of *R'G'B'* values to the corresponding *YCoCg* colour encoding values array.
YCoCg_to_RGB(YCoCg)	Converts an array of *YCoCg* colour encoding values to the corresponding *R'G'B'* values array.

colour.RGB_to_YCoCg

colour.**RGB_to_YCoCg**(*RGB*)

Converts an array of *R'G'B'* values to the corresponding *YCoCg* colour encoding values array.

Parameters **RGB** (array_like) – Input *R'G'B'* array.

Returns *YCoCg* colour encoding array.

Return type ndarray

References

[MS03]

Examples

```
>>> RGB_to_YCoCg(np.array([1.0, 1.0, 1.0]))
array([ 1.,  0.,  0.])
>>> RGB_to_YCoCg(np.array([0.75, 0.5, 0.5]))
array([ 0.5625,  0.125 , -0.0625])
```

colour.YCoCg_to_RGB

colour.**YCoCg_to_RGB**(*YCoCg*)

Converts an array of *YCoCg* colour encoding values to the corresponding *R'G'B'* values array.

Parameters **YCoCg** (array_like) – *YCoCg* colour encoding array.

Returns Output *R'G'B'* array.

Return type ndarray

References

[MS03]

Examples

```
>>> YCoCg_to_RGB(np.array([1.0, 0.0, 0.0]))
array([ 1.,  1.,  1.])
>>> YCoCg_to_RGB(np.array([0.5625, 0.125, -0.0625]))
array([ 0.75,  0.5 ,  0.5 ])
```

IC_TC_P Colour Encoding

colour

RGB_to_ICTCP(RGB[, L_p])	Converts from *ITU-R BT.2020* colourspace to IC_TC_P colour encoding.
ICTCP_to_RGB(ICTCP[, L_p])	Converts from IC_TC_P colour encoding to *ITU-R BT.2020* colourspace.

colour.RGB_to_ICTCP

colour.**RGB_to_ICTCP**(*RGB*, *L_p=10000*)

Converts from *ITU-R BT.2020* colourspace to IC_TC_P colour encoding.

Parameters

- **RGB** (array_like) – *ITU-R BT.2020* colourspace array.
- **L_p** (numeric, optional) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding.

Returns IC_TC_P colour encoding array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
ICTCP	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

References

[Dol16], [LPY+16]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_ICTCP(RGB)
array([ 0.0735136...,  0.0047525...,  0.0935159...])
```

colour.ICTCP_to_RGB

colour.**ICTCP_to_RGB**(*ICTCP*, *L_p=10000*)

Converts from IC_TC_P colour encoding to *ITU-R BT.2020* colourspace.

Parameters

- **ICTCP** (array_like) – IC_TC_P colour encoding array.
- **L_p** (numeric, optional) – Display peak luminance cd/m^2 for *SMPTE ST 2084:2014* non-linear encoding.

Returns *ITU-R BT.2020* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
ICTCP	I : [0, 1] CT : [-1, 1] CP : [-1, 1]	I : [0, 1] CT : [-1, 1] CP : [-1, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Dol16], [LPY+16]

Examples

```
>>> ICTCP = np.array([0.07351364, 0.00475253, 0.09351596])
>>> ICTCP_to_RGB(ICTCP)
array([ 0.4562052...,  0.0308107...,  0.0409195...])
```

RGB Representations

Prismatic Colourspace

colour

RGB_to_Prismatic(RGB)	Converts from *RGB* colourspace to *Prismatic* $L\rho\gamma\beta$ colourspace array.
Prismatic_to_RGB(Lrgb)	Converts from *Prismatic* $L\rho\gamma\beta$ colourspace array to *RGB* colourspace.

colour.RGB_to_Prismatic

colour.**RGB_to_Prismatic**(*RGB*)

Converts from *RGB* colourspace to *Prismatic* $L\rho\gamma\beta$ colourspace array.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns *Prismatic* $L\rho\gamma\beta$ colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
Lrgb	[0, 1]	[0, 1]

References

[SH15]

Examples

```
>>> RGB = np.array([0.25, 0.50, 0.75])
>>> RGB_to_Prismatic(RGB)
array([ 0.75...   ,  0.1666666...,  0.3333333...,  0.5...   ])
```

Adjusting saturation of given *RGB* colourspace array: >>> saturation = 0.5 >>> Lrgb = RGB_to_Prismatic(RGB) >>> Lrgb[..., 1:] = 1 / 3 + saturation * (Lrgb[..., 1:] - 1 / 3) >>> Prismatic_to_RGB(Lrgb) # doctest: +ELLIPSIS array([0.45..., 0.6..., 0.75...])

colour.Prismatic_to_RGB

colour.**Prismatic_to_RGB**(*Lrgb*)

Converts from *Prismatic* $L\rho\gamma\beta$ colourspace array to *RGB* colourspace.

Parameters **Lrgb** (array_like) – *Prismatic* $L\rho\gamma\beta$ colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
Lrgb	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[SH15]

Examples

```
>>> Lrgb = np.array([0.75000000, 0.16666667, 0.33333333, 0.50000000])
>>> Prismatic_to_RGB(Lrgb)
array([ 0.25...   ,  0.4999999...,  0.75...  ])
```

HSV Colourspace

colour

RGB_to_HSV(RGB)	Converts from *RGB* colourspace to *HSV* colourspace.
HSV_to_RGB(HSV)	Converts from *HSV* colourspace to *RGB* colourspace.

colour.RGB_to_HSV

colour.**RGB_to_HSV**(*RGB*)

Converts from *RGB* colourspace to *HSV* colourspace.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns *HSV* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HSV	[0, 1]	[0, 1]

References

[Eash], [Smi78], [Wik03a]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSV(RGB)
array([ 0.9960394...,  0.9324630...,  0.4562051...])
```

colour.HSV_to_RGB

colour.**HSV_to_RGB**(*HSV*)

Converts from *HSV* colourspace to *RGB* colourspace.

Parameters **HSV** (array_like) – *HSV* colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
HSV	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Ease], [Smi78], [Wik03a]

Examples

```
>>> HSV = np.array([0.99603944, 0.93246304, 0.45620519])
>>> HSV_to_RGB(HSV)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

HSL Colourspace

colour

RGB_to_HSL(RGB)	Converts from *RGB* colourspace to *HSL* colourspace.
HSL_to_RGB(HSL)	Converts from *HSL* colourspace to *RGB* colourspace.

colour.RGB_to_HSL

colour.**RGB_to_HSL**(*RGB*)

Converts from *RGB* colourspace to *HSL* colourspace.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns *HSL* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
HSL	[0, 1]	[0, 1]

References

[Easg], [Smi78], [Wik03a]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_HSL(RGB)
array([ 0.9960394...,  0.8734714...,  0.2435079...])
```

colour.HSL_to_RGB

colour.**HSL_to_RGB**(*HSL*)

Converts from *HSL* colourspace to *RGB* colourspace.

Parameters **HSL** (array_like) – *HSL* colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
HSL	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Easd], [Smi78], [Wik03a]

Examples

```
>>> HSL = np.array([0.99603944, 0.87347144, 0.24350795])
>>> HSL_to_RGB(HSL)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

CMY Colourspace

colour

RGB_to_CMY(RGB)	Converts from *RGB* colourspace to *CMY* colourspace.
CMY_to_RGB(CMY)	Converts from *CMY* colourspace to *CMY* colourspace.
CMY_to_CMYK(CMY)	Converts from *CMY* colourspace to *CMYK* colourspace.
CMYK_to_CMY(CMYK)	Converts from *CMYK* colourspace to *CMY* colourspace.

colour.RGB_to_CMY

colour.**RGB_to_CMY**(*RGB*)

Converts from *RGB* colourspace to *CMY* colourspace.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns *CMY* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

References

[Easf]

Examples

```
>>> RGB = np.array([0.45620519, 0.03081071, 0.04091952])
>>> RGB_to_CMY(RGB)
array([ 0.5437948...,  0.9691892...,  0.9590804...])
```

colour.CMY_to_RGB

colour.**CMY_to_RGB**(*CMY*)

Converts from *CMY* colourspace to *CMY* colourspace.

Parameters **CMY** (array_like) – *CMY* colourspace array.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Easb]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_RGB(CMY)
array([ 0.4562051...,  0.0308107...,  0.0409195...])
```

colour.CMY_to_CMYK

colour.**CMY_to_CMYK**(*CMY*)

Converts from *CMY* colourspace to *CMYK* colourspace.

Parameters **CMY** (array_like) – *CMY* colourspace array.

Returns *CMYK* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMYK	[0, 1]	[0, 1]

References

[Easa]

Examples

```
>>> CMY = np.array([0.54379481, 0.96918929, 0.95908048])
>>> CMY_to_CMYK(CMY)
array([ 0.        ,  0.9324630...,  0.9103045...,  0.5437948...])
```

colour.CMYK_to_CMY

colour.**CMYK_to_CMY**(*CMYK*)

Converts from *CMYK* colourspace to *CMY* colourspace.

Parameters **CMYK** (array_like) – *CMYK* colourspace array.

Returns *CMY* array.

Return type ndarray

Notes

Domain	Scale - Reference	Scale - 1
CMYK	[0, 1]	[0, 1]

Range	Scale - Reference	Scale - 1
CMY	[0, 1]	[0, 1]

References

[Easc]

Examples

```
>>> CMYK = np.array([0.50000000, 0.00000000, 0.74400000, 0.01960784])
>>> CMYK_to_CMY(CMYK)
array([ 0.5098039...,  0.0196078...,  0.7490196...])
```

Pointer's Gamut

colour

POINTER_GAMUT_BOUNDARIES
POINTER_GAMUT_DATA
POINTER_GAMUT_ILLUMINANT

colour.POINTER_GAMUT_BOUNDARIES

colour.**POINTER_GAMUT_BOUNDARIES = array([[0.659, 0.316], [0.634, 0.351], [0.594, 0.391], [0.557, 0.42**

colour.POINTER_GAMUT_DATA

colour.**POINTER_GAMUT_DATA = array([[15, 10, 0], [15, 15, 10], [15, 14, 20], ..., [90, 9, 330], [90,**

colour.POINTER_GAMUT_ILLUMINANT

colour.**POINTER_GAMUT_ILLUMINANT = array([0.31005673, 0.3161457])**

Colour Notation Systems

- *Munsell Renotation System*
- *Munsell Value*
  - *Priest, Gibson and MacNicholas (1920)*
  - *Munsell, Sloan and Godlove (1933)*

- *Moon and Spencer (1943)*
- *Saunderson and Milner (1944)*
- *Ladd and Pinney (1955)*
- *McCamy (1987)*
- *ASTM D1535-08e1*

- *Hexadecimal Triplet Notation*

Munsell Renotation System

colour

munsell_colour_to_xyY(munsell_colour)	Converts given *Munsell* colour to *CIE xyY* colourspace.
xyY_to_munsell_colour(xyY[, hue_decimals, ...])	Converts from *CIE xyY* colourspace to *Munsell* colour.

colour.munsell_colour_to_xyY

colour.**munsell_colour_to_xyY**(*munsell_colour*)

Converts given *Munsell* colour to *CIE xyY* colourspace.

Parameters **munsell_colour** (unicode or array_like) – *Munsell* colour.

Returns *CIE xyY* colourspace array.

Return type ndarray

Notes

Range	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Cen], [Cen12]

Examples

```
>>> munsell_colour_to_xyY('4.2YR 8.1/5.3')
array([ 0.3873694...,  0.3575165...,  0.59362  ])
>>> munsell_colour_to_xyY('N8.9')
array([ 0.31006  ,  0.31616  ,  0.7461345...])
```

colour.xyY_to_munsell_colour

colour.**xyY_to_munsell_colour**(*xyY*, *hue_decimals=1*, *value_decimals=1*, *chroma_decimals=1*)

Converts from *CIE xyY* colourspace to *Munsell* colour.

Parameters

- **xyY** (array_like, (3,)) – *CIE xyY* colourspace array.
- **hue_decimals** (int) – Hue formatting decimals.
- **value_decimals** (int) – Value formatting decimals.
- **chroma_decimals** (int) – Chroma formatting decimals.

Returns *Munsell* colour.

Return type unicode

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

References

[Cen], [Cen12]

Examples

```
>>> xyY = np.array([0.38736945, 0.35751656, 0.59362000])
>>> # Doctests skip for Python 2.x compatibility.
>>> xyY_to_munsell_colour(xyY)
'4.2YR 8.1/5.3'
```

Dataset

colour

MUNSELL_COLOURS	Aggregated *Munsell* colours.

colour.MUNSELL_COLOURS

colour.**MUNSELL_COLOURS = CaseInsensitiveMapping({'Munsell Colours All': ..., 'Munsell Colours 1929': ...**

Aggregated *Munsell* colours.

MUNSELL_COLOURS : CaseInsensitiveMapping

Aliases:

- ‘all’: ‘Munsell Colours All’
- ‘1929’: ‘Munsell Colours 1929’
- ‘real’: ‘Munsell Colours Real’

Munsell Value

colour

munsell_value(Y[, method])	Returns the *Munsell* value V of given *luminance* Y using given method.
MUNSELL_VALUE_METHODS	Supported *Munsell* value computation methods.

colour.munsell_value

colour.**munsell_value**(*Y*, *method='ASTM D1535'*)

Returns the *Munsell* value V of given *luminance* Y using given method.

Parameters

- **Y** (numeric or array_like) – *luminance* Y.
- **method** (unicode, optional) – **{'ASTM D1535', 'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987'}**, Computation method.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[ASTMInternational89], [Wik07d]

Examples

```
>>> munsell_value(12.23634268)
4.0824437...
>>> munsell_value(12.23634268, method='Priest 1920')
3.4980484...
>>> munsell_value(12.23634268, method='Munsell 1933')
4.1627702...
>>> munsell_value(12.23634268, method='Moon 1943')
4.0688120...
>>> munsell_value(12.23634268, method='Saunderson 1944')
...
4.0444736...
>>> munsell_value(12.23634268, method='Ladd 1955')
4.0511633...
>>> munsell_value(12.23634268, method='McCamy 1987')
array(4.0814348...)
```

colour.MUNSELL_VALUE_METHODS

colour.**MUNSELL_VALUE_METHODS = CaseInsensitiveMapping({'Priest 1920': ..., 'Munsell 1933': ..., 'Moon 19**
Supported *Munsell* value computation methods.

References

[ASTMInternational89], [Wik07d]

MUNSELL_VALUE_METHODS [CaseInsensitiveMapping] **{'Priest 1920', 'Munsell 1933', 'Moon 1943', 'Saunderson 1944', 'Ladd 1955', 'McCamy 1987', 'ASTM D1535'}**

Aliases:

- 'astm2008': 'ASTM D1535'

Priest, Gibson and MacNicholas (1920)

colour.notation

munsell_value_Priest1920(Y)	Returns the *Munsell* value V of given *luminance* Y using *Priest et al.(1920)* method..

colour.notation.munsell_value_Priest1920

colour.notation.**munsell_value_Priest1920**(Y)
Returns the *Munsell* value V of given *luminance* Y using *Priest et al. (1920)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Priest1920(12.23634268)
3.4980484...
```

Munsell, Sloan and Godlove (1933)

colour.notation

munsell_value_Munsell1933(Y)	Returns the *Munsell* value V of given *luminance* Y using *Munsell et al.(1933)* method..

colour.notation.munsell_value_Munsell1933

colour.notation.**munsell_value_Munsell1933**(Y)

Returns the *Munsell* value V of given *luminance* Y using *Munsell et al. (1933)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Munsell1933(12.23634268)
4.1627702...
```

Moon and Spencer (1943)

colour.notation

munsell_value_Moon1943(Y)	Returns the *Munsell* value V of given *luminance* Y using *Moon and Spencer (1943)* method.

colour.notation.munsell_value_Moon1943

colour.notation.**munsell_value_Moon1943**(*Y*)

Returns the *Munsell* value V of given *luminance* Y using *Moon and Spencer (1943)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Moon1943(12.23634268)
4.0688120...
```

Saunderson and Milner (1944)

colour.notation

munsell_value_Saunderson1944(Y)	Returns the *Munsell* value V of given *luminance* Y using *Saunderson and Milner (1944)* method.

colour.notation.munsell_value_Saunderson1944

colour.notation.**munsell_value_Saunderson1944**(*Y*)

Returns the *Munsell* value V of given *luminance* Y using *Saunderson and Milner (1944)* method.

Parameters **Y** (numeric) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Saunderson1944(12.23634268)
4.0444736...
```

Ladd and Pinney (1955)

`colour.notation`

`munsell_value_Ladd1955`(Y)	Returns the *Munsell* value V of given *luminance* Y using *Ladd and Pinney (1955)* method.

colour.notation.munsell_value_Ladd1955

`colour.notation.`**`munsell_value_Ladd1955`**(Y)

Returns the *Munsell* value V of given *luminance* Y using *Ladd and Pinney (1955)* method.

Parameters **Y** (`numeric or array_like`) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[Wik07d]

Examples

```
>>> munsell_value_Ladd1955(12.23634268)
4.0511633...
```

McCamy (1987)

colour.notation

munsell_value_McCamy1987(Y)	Returns the *Munsell* value V of given *luminance* Y using *McCamy (1987)* method.

colour.notation.munsell_value_McCamy1987

colour.notation.**munsell_value_McCamy1987**(Y)

Returns the *Munsell* value V of given *luminance* Y using *McCamy (1987)* method.

Parameters **Y** (numeric or array_like) – *luminance* Y.

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

References

[ASTMInternational89]

Examples

```
>>> munsell_value_McCamy1987(12.23634268)
array(4.0814348...)
```

ASTM D1535-08e1

colour.notation

munsell_value_ASTMD1535(Y)	Returns the *Munsell* value V of given *luminance* Y using an inverse lookup table from *ASTM D1535-08e1* method.

colour.notation.munsell_value_ASTMD1535

colour.notation.**munsell_value_ASTMD1535**(Y)

Returns the *Munsell* value V of given *luminance* Y using an inverse lookup table from *ASTM D1535-08e1* method.

Parameters **Y** (numeric or array_like) – *luminance* Y

Returns *Munsell* value V.

Return type numeric or ndarray

Notes

Domain	Scale - Reference	Scale - 1
Y	[0, 100]	[0, 1]

Range	Scale - Reference	Scale - 1
V	[0, 10]	[0, 1]

- The *Munsell* value* computation with *ASTM D1535-08e1* method is only defined for domain [0, 100].

References

[ASTMInternational89]

Examples

```
>>> munsell_value_ASTMD1535(12.23634268)
4.0824437...
```

Hexadecimal Triplet Notation

colour.notation

RGB_to_HEX(RGB)	Converts from *RGB* colourspace to hexadecimal triplet representation.
HEX_to_RGB(HEX)	Converts from hexadecimal triplet representation to *RGB* colourspace.

colour.notation.RGB_to_HEX

colour.notation.**RGB_to_HEX**(*RGB*)

Converts from *RGB* colourspace to hexadecimal triplet representation.

Parameters **RGB** (array_like) – *RGB* colourspace array.

Returns Hexadecimal triplet representation.

Return type unicode

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> RGB = np.array([0.66666667, 0.86666667, 1.00000000])
>>> # Doctests skip for Python 2.x compatibility.
>>> RGB_to_HEX(RGB)
'#aaddff'
```

colour.notation.HEX_to_RGB

colour.notation.**HEX_to_RGB**(*HEX*)

Converts from hexadecimal triplet representation to *RGB* colourspace.

Parameters **HEX** (unicode or array_like) – Hexadecimal triplet representation.

Returns *RGB* colourspace array.

Return type ndarray

Notes

Notes

Range	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

Examples

```
>>> HEX = '#aaddff'
>>> HEX_to_RGB(HEX)
array([ 0.6666666...,  0.8666666...,  1.        ])
```

Optical Phenomena

- *Rayleigh Scattering*

Rayleigh Scattering

colour

rayleigh_scattering(wavelength[, ...])	Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).
sd_rayleigh_scattering([shape, ...])	Returns the *Rayleigh* spectral distribution for given spectral shape.
scattering_cross_section(wavelength[, ...])	Returns the scattering cross section per molecule σ of dry air as function of wavelength λ in centimeters (cm) using given CO_2 concentration in parts per million (ppm) and temperature $T[K]$ in kelvin degrees following *Van de Hulst (1957)* method.

colour.rayleigh_scattering

colour.**rayleigh_scattering**(*wavelength*, *CO2_concentration=300*, *temperature=288.15*, *pressure=101325*, *latitude=0*, *altitude=0*, *avogadro_constant=6.02214179e+23*, *n_s=<function air_refraction_index_Bodhaine1999>*, *F_air=<function F_air_Bodhaine1999>*)

Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric or array_like) – Surface pressure P of the measurement site.
- **latitude** (numeric or array_like, optional) – Latitude of the site in degrees.
- **altitude** (numeric or array_like, optional) – Altitude of the site in meters.
- **avogadro_constant** (numeric or array_like, optional) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6+3_p)/(6-7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth $T_r(\lambda)$.

Return type numeric or ndarray

Warning: Unlike most objects of `colour.phenomena.rayleigh` module, `colour.phenomena.rayleigh_optical_depth()` expects wavelength λ to be expressed in centimeters (cm).

References

[BWDS99], [Wik01d]

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
0.1004070...
```

colour.sd_rayleigh_scattering

colour.**sd_rayleigh_scattering**(*shape=SpectralShape(360, 780, 1)*, *CO2_concentration=300*, *temperature=288.15*, *pressure=101325*, *latitude=0*, *altitude=0*, *avogadro_constant=6.02214179e+23*, *n_s=<function air_refraction_index_Bodhaine1999>*, *F_air=<function F_air_Bodhaine1999>*)

Returns the *Rayleigh* spectral distribution for given spectral shape.

Parameters

- **shape** (`SpectralShape, optional`) – Spectral shape used to create the *Rayleigh* scattering spectral distribution.
- **CO2_concentration** (`numeric or array_like, optional`) – CO_2 concentration in parts per million (ppm).
- **temperature** (`numeric or array_like, optional`) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (`numeric or array_like`) – Surface pressure P of the measurement site.
- **latitude** (`numeric or array_like, optional`) – Latitude of the site in degrees.
- **altitude** (`numeric or array_like, optional`) – Altitude of the site in meters.
- **avogadro_constant** (`numeric or array_like, optional`) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s** (`object`) – Air refraction index n_s computation method.
- **F_air** (`object`) – $(6+3_p)/(6-7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth spectral distribution.

Return type *SpectralDistribution*

References

[BWDS99], [Wik01d]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> with numpy_print_options(suppress=True):
...     sd_rayleigh_scattering()
SpectralDistribution([[ 360.        ,    0.5991013...],
                      [ 361.        ,    0.5921706...],
                      [ 362.        ,    0.5853410...],
                      [ 363.        ,    0.5786105...],
                      [ 364.        ,    0.5719774...],
                      [ 365.        ,    0.5654401...],
                      [ 366.        ,    0.5589968...],
                      [ 367.        ,    0.5526460...],
                      [ 368.        ,    0.5463860...],
                      [ 369.        ,    0.5402153...],
                      [ 370.        ,    0.5341322...],
                      [ 371.        ,    0.5281354...],
                      [ 372.        ,    0.5222234...],
                      [ 373.        ,    0.5163946...],
                      [ 374.        ,    0.5106476...],
                      [ 375.        ,    0.5049812...],
                      [ 376.        ,    0.4993939...],
                      [ 377.        ,    0.4938844...],
                      [ 378.        ,    0.4884513...],
                      [ 379.        ,    0.4830934...],
                      [ 380.        ,    0.4778095...],
                      [ 381.        ,    0.4725983...],
                      [ 382.        ,    0.4674585...],
                      [ 383.        ,    0.4623891...],
                      [ 384.        ,    0.4573889...],
                      [ 385.        ,    0.4524566...],
                      [ 386.        ,    0.4475912...],
                      [ 387.        ,    0.4427917...],
                      [ 388.        ,    0.4380568...],
                      [ 389.        ,    0.4333856...],
                      [ 390.        ,    0.4287771...],
                      [ 391.        ,    0.4242302...],
                      [ 392.        ,    0.4197439...],
                      [ 393.        ,    0.4153172...],
                      [ 394.        ,    0.4109493...],
                      [ 395.        ,    0.4066391...],
                      [ 396.        ,    0.4023857...],
                      [ 397.        ,    0.3981882...],
                      [ 398.        ,    0.3940458...],
                      [ 399.        ,    0.3899576...],
                      [ 400.        ,    0.3859227...],
                      [ 401.        ,    0.3819402...],
                      [ 402.        ,    0.3780094...],
                      [ 403.        ,    0.3741295...],
                      [ 404.        ,    0.3702996...],
                      [ 405.        ,    0.366519 ...],
                      [ 406.        ,    0.3627868...],
                      [ 407.        ,    0.3591025...],
                      [ 408.        ,    0.3554651...],
                      [ 409.        ,    0.3518740...],
                      [ 410.        ,    0.3483286...],
                      [ 411.        ,    0.344828 ...],
```

(continues on next page)

(continued from previous page)

```
                    [ 412.        ,    0.3413716...],
                    [ 413.        ,    0.3379587...],
                    [ 414.        ,    0.3345887...],
                    [ 415.        ,    0.3312609...],
                    [ 416.        ,    0.3279747...],
                    [ 417.        ,    0.3247294...],
                    [ 418.        ,    0.3215245...],
                    [ 419.        ,    0.3183593...],
                    [ 420.        ,    0.3152332...],
                    [ 421.        ,    0.3121457...],
                    [ 422.        ,    0.3090962...],
                    [ 423.        ,    0.3060841...],
                    [ 424.        ,    0.3031088...],
                    [ 425.        ,    0.3001699...],
                    [ 426.        ,    0.2972668...],
                    [ 427.        ,    0.2943989...],
                    [ 428.        ,    0.2915657...],
                    [ 429.        ,    0.2887668...],
                    [ 430.        ,    0.2860017...],
                    [ 431.        ,    0.2832697...],
                    [ 432.        ,    0.2805706...],
                    [ 433.        ,    0.2779037...],
                    [ 434.        ,    0.2752687...],
                    [ 435.        ,    0.2726650...],
                    [ 436.        ,    0.2700922...],
                    [ 437.        ,    0.2675500...],
                    [ 438.        ,    0.2650377...],
                    [ 439.        ,    0.2625551...],
                    [ 440.        ,    0.2601016...],
                    [ 441.        ,    0.2576770...],
                    [ 442.        ,    0.2552807...],
                    [ 443.        ,    0.2529124...],
                    [ 444.        ,    0.2505716...],
                    [ 445.        ,    0.2482581...],
                    [ 446.        ,    0.2459713...],
                    [ 447.        ,    0.2437110...],
                    [ 448.        ,    0.2414768...],
                    [ 449.        ,    0.2392683...],
                    [ 450.        ,    0.2370851...],
                    [ 451.        ,    0.2349269...],
                    [ 452.        ,    0.2327933...],
                    [ 453.        ,    0.2306841...],
                    [ 454.        ,    0.2285989...],
                    [ 455.        ,    0.2265373...],
                    [ 456.        ,    0.2244990...],
                    [ 457.        ,    0.2224838...],
                    [ 458.        ,    0.2204912...],
                    [ 459.        ,    0.2185211...],
                    [ 460.        ,    0.2165730...],
                    [ 461.        ,    0.2146467...],
                    [ 462.        ,    0.2127419...],
                    [ 463.        ,    0.2108583...],
                    [ 464.        ,    0.2089957...],
                    [ 465.        ,    0.2071536...],
                    [ 466.        ,    0.2053320...],
                    [ 467.        ,    0.2035304...],
                    [ 468.        ,    0.2017487...],
                    [ 469.        ,    0.1999865...],
                    [ 470.        ,    0.1982436...],
                    [ 471.        ,    0.1965198...],
                    [ 472.        ,    0.1948148...],
```

(continues on next page)

(continued from previous page)

```
                       [ 473.        ,    0.1931284...],
                       [ 474.        ,    0.1914602...],
                       [ 475.        ,    0.1898101...],
                       [ 476.        ,    0.1881779...],
                       [ 477.        ,    0.1865633...],
                       [ 478.        ,    0.1849660...],
                       [ 479.        ,    0.1833859...],
                       [ 480.        ,    0.1818227...],
                       [ 481.        ,    0.1802762...],
                       [ 482.        ,    0.1787463...],
                       [ 483.        ,    0.1772326...],
                       [ 484.        ,    0.1757349...],
                       [ 485.        ,    0.1742532...],
                       [ 486.        ,    0.1727871...],
                       [ 487.        ,    0.1713365...],
                       [ 488.        ,    0.1699011...],
                       [ 489.        ,    0.1684809...],
                       [ 490.        ,    0.1670755...],
                       [ 491.        ,    0.1656848...],
                       [ 492.        ,    0.1643086...],
                       [ 493.        ,    0.1629468...],
                       [ 494.        ,    0.1615991...],
                       [ 495.        ,    0.1602654...],
                       [ 496.        ,    0.1589455...],
                       [ 497.        ,    0.1576392...],
                       [ 498.        ,    0.1563464...],
                       [ 499.        ,    0.1550668...],
                       [ 500.        ,    0.1538004...],
                       [ 501.        ,    0.1525470...],
                       [ 502.        ,    0.1513063...],
                       [ 503.        ,    0.1500783...],
                       [ 504.        ,    0.1488628...],
                       [ 505.        ,    0.1476597...],
                       [ 506.        ,    0.1464687...],
                       [ 507.        ,    0.1452898...],
                       [ 508.        ,    0.1441228...],
                       [ 509.        ,    0.1429675...],
                       [ 510.        ,    0.1418238...],
                       [ 511.        ,    0.1406916...],
                       [ 512.        ,    0.1395707...],
                       [ 513.        ,    0.1384610...],
                       [ 514.        ,    0.1373624...],
                       [ 515.        ,    0.1362747...],
                       [ 516.        ,    0.1351978...],
                       [ 517.        ,    0.1341316...],
                       [ 518.        ,    0.1330759...],
                       [ 519.        ,    0.1320306...],
                       [ 520.        ,    0.1309956...],
                       [ 521.        ,    0.1299707...],
                       [ 522.        ,    0.1289559...],
                       [ 523.        ,    0.1279511...],
                       [ 524.        ,    0.1269560...],
                       [ 525.        ,    0.1259707...],
                       [ 526.        ,    0.1249949...],
                       [ 527.        ,    0.1240286...],
                       [ 528.        ,    0.1230717...],
                       [ 529.        ,    0.1221240...],
                       [ 530.        ,    0.1211855...],
                       [ 531.        ,    0.1202560...],
                       [ 532.        ,    0.1193354...],
                       [ 533.        ,    0.1184237...],
```

(continues on next page)

(continued from previous page)

```
                [ 534.        ,    0.1175207...],
                [ 535.        ,    0.1166263...],
                [ 536.        ,    0.1157404...],
                [ 537.        ,    0.1148630...],
                [ 538.        ,    0.1139939...],
                [ 539.        ,    0.1131331...],
                [ 540.        ,    0.1122804...],
                [ 541.        ,    0.1114357...],
                [ 542.        ,    0.1105990...],
                [ 543.        ,    0.1097702...],
                [ 544.        ,    0.1089492...],
                [ 545.        ,    0.1081358...],
                [ 546.        ,    0.1073301...],
                [ 547.        ,    0.1065319...],
                [ 548.        ,    0.1057411...],
                [ 549.        ,    0.1049577...],
                [ 550.        ,    0.1041815...],
                [ 551.        ,    0.1034125...],
                [ 552.        ,    0.1026507...],
                [ 553.        ,    0.1018958...],
                [ 554.        ,    0.1011480...],
                [ 555.        ,    0.1004070...],
                [ 556.        ,    0.0996728...],
                [ 557.        ,    0.0989453...],
                [ 558.        ,    0.0982245...],
                [ 559.        ,    0.0975102...],
                [ 560.        ,    0.0968025...],
                [ 561.        ,    0.0961012...],
                [ 562.        ,    0.0954062...],
                [ 563.        ,    0.0947176...],
                [ 564.        ,    0.0940352...],
                [ 565.        ,    0.0933589...],
                [ 566.        ,    0.0926887...],
                [ 567.        ,    0.0920246...],
                [ 568.        ,    0.0913664...],
                [ 569.        ,    0.0907141...],
                [ 570.        ,    0.0900677...],
                [ 571.        ,    0.0894270...],
                [ 572.        ,    0.0887920...],
                [ 573.        ,    0.0881627...],
                [ 574.        ,    0.0875389...],
                [ 575.        ,    0.0869207...],
                [ 576.        ,    0.0863079...],
                [ 577.        ,    0.0857006...],
                [ 578.        ,    0.0850986...],
                [ 579.        ,    0.0845019...],
                [ 580.        ,    0.0839104...],
                [ 581.        ,    0.0833242...],
                [ 582.        ,    0.0827430...],
                [ 583.        ,    0.082167 ...],
                [ 584.        ,    0.0815959...],
                [ 585.        ,    0.0810298...],
                [ 586.        ,    0.0804687...],
                [ 587.        ,    0.0799124...],
                [ 588.        ,    0.0793609...],
                [ 589.        ,    0.0788142...],
                [ 590.        ,    0.0782722...],
                [ 591.        ,    0.0777349...],
                [ 592.        ,    0.0772022...],
                [ 593.        ,    0.0766740...],
                [ 594.        ,    0.0761504...],
```

(continues on next page)

(continued from previous page)

```
                    [ 595.        ,    0.0756313...],
                    [ 596.        ,    0.0751166...],
                    [ 597.        ,    0.0746063...],
                    [ 598.        ,    0.0741003...],
                    [ 599.        ,    0.0735986...],
                    [ 600.        ,    0.0731012...],
                    [ 601.        ,    0.072608 ...],
                    [ 602.        ,    0.0721189...],
                    [ 603.        ,    0.0716340...],
                    [ 604.        ,    0.0711531...],
                    [ 605.        ,    0.0706763...],
                    [ 606.        ,    0.0702035...],
                    [ 607.        ,    0.0697347...],
                    [ 608.        ,    0.0692697...],
                    [ 609.        ,    0.0688087...],
                    [ 610.        ,    0.0683515...],
                    [ 611.        ,    0.0678981...],
                    [ 612.        ,    0.0674485...],
                    [ 613.        ,    0.0670026...],
                    [ 614.        ,    0.0665603...],
                    [ 615.        ,    0.0661218...],
                    [ 616.        ,    0.0656868...],
                    [ 617.        ,    0.0652555...],
                    [ 618.        ,    0.0648277...],
                    [ 619.        ,    0.0644033...],
                    [ 620.        ,    0.0639825...],
                    [ 621.        ,    0.0635651...],
                    [ 622.        ,    0.0631512...],
                    [ 623.        ,    0.0627406...],
                    [ 624.        ,    0.0623333...],
                    [ 625.        ,    0.0619293...],
                    [ 626.        ,    0.0615287...],
                    [ 627.        ,    0.0611312...],
                    [ 628.        ,    0.0607370...],
                    [ 629.        ,    0.0603460...],
                    [ 630.        ,    0.0599581...],
                    [ 631.        ,    0.0595733...],
                    [ 632.        ,    0.0591917...],
                    [ 633.        ,    0.0588131...],
                    [ 634.        ,    0.0584375...],
                    [ 635.        ,    0.0580649...],
                    [ 636.        ,    0.0576953...],
                    [ 637.        ,    0.0573286...],
                    [ 638.        ,    0.0569649...],
                    [ 639.        ,    0.0566040...],
                    [ 640.        ,    0.0562460...],
                    [ 641.        ,    0.0558909...],
                    [ 642.        ,    0.0555385...],
                    [ 643.        ,    0.0551890...],
                    [ 644.        ,    0.0548421...],
                    [ 645.        ,    0.0544981...],
                    [ 646.        ,    0.0541567...],
                    [ 647.        ,    0.053818 ...],
                    [ 648.        ,    0.0534819...],
                    [ 649.        ,    0.0531485...],
                    [ 650.        ,    0.0528176...],
                    [ 651.        ,    0.0524894...],
                    [ 652.        ,    0.0521637...],
                    [ 653.        ,    0.0518405...],
                    [ 654.        ,    0.0515198...],
                    [ 655.        ,    0.0512017...],
```

(continues on next page)

(continued from previous page)

```
                       [ 656.        ,    0.0508859...],
                       [ 657.        ,    0.0505726...],
                       [ 658.        ,    0.0502618...],
                       [ 659.        ,    0.0499533...],
                       [ 660.        ,    0.0496472...],
                       [ 661.        ,    0.0493434...],
                       [ 662.        ,    0.0490420...],
                       [ 663.        ,    0.0487428...],
                       [ 664.        ,    0.0484460...],
                       [ 665.        ,    0.0481514...],
                       [ 666.        ,    0.0478591...],
                       [ 667.        ,    0.0475689...],
                       [ 668.        ,    0.0472810...],
                       [ 669.        ,    0.0469953...],
                       [ 670.        ,    0.0467117...],
                       [ 671.        ,    0.0464302...],
                       [ 672.        ,    0.0461509...],
                       [ 673.        ,    0.0458737...],
                       [ 674.        ,    0.0455986...],
                       [ 675.        ,    0.0453255...],
                       [ 676.        ,    0.0450545...],
                       [ 677.        ,    0.0447855...],
                       [ 678.        ,    0.0445185...],
                       [ 679.        ,    0.0442535...],
                       [ 680.        ,    0.0439905...],
                       [ 681.        ,    0.0437294...],
                       [ 682.        ,    0.0434703...],
                       [ 683.        ,    0.0432131...],
                       [ 684.        ,    0.0429578...],
                       [ 685.        ,    0.0427044...],
                       [ 686.        ,    0.0424529...],
                       [ 687.        ,    0.0422032...],
                       [ 688.        ,    0.0419553...],
                       [ 689.        ,    0.0417093...],
                       [ 690.        ,    0.0414651...],
                       [ 691.        ,    0.0412226...],
                       [ 692.        ,    0.0409820...],
                       [ 693.        ,    0.0407431...],
                       [ 694.        ,    0.0405059...],
                       [ 695.        ,    0.0402705...],
                       [ 696.        ,    0.0400368...],
                       [ 697.        ,    0.0398047...],
                       [ 698.        ,    0.0395744...],
                       [ 699.        ,    0.0393457...],
                       [ 700.        ,    0.0391187...],
                       [ 701.        ,    0.0388933...],
                       [ 702.        ,    0.0386696...],
                       [ 703.        ,    0.0384474...],
                       [ 704.        ,    0.0382269...],
                       [ 705.        ,    0.0380079...],
                       [ 706.        ,    0.0377905...],
                       [ 707.        ,    0.0375747...],
                       [ 708.        ,    0.0373604...],
                       [ 709.        ,    0.0371476...],
                       [ 710.        ,    0.0369364...],
                       [ 711.        ,    0.0367266...],
                       [ 712.        ,    0.0365184...],
                       [ 713.        ,    0.0363116...],
                       [ 714.        ,    0.0361063...],
                       [ 715.        ,    0.0359024...],
                       [ 716.        ,    0.0357000...],
```

(continues on next page)

(continued from previous page)

```
                       [ 717.        ,    0.0354990...],
                       [ 718.        ,    0.0352994...],
                       [ 719.        ,    0.0351012...],
                       [ 720.        ,    0.0349044...],
                       [ 721.        ,    0.0347090...],
                       [ 722.        ,    0.0345150...],
                       [ 723.        ,    0.0343223...],
                       [ 724.        ,    0.0341310...],
                       [ 725.        ,    0.0339410...],
                       [ 726.        ,    0.0337523...],
                       [ 727.        ,    0.033565 ...],
                       [ 728.        ,    0.0333789...],
                       [ 729.        ,    0.0331941...],
                       [ 730.        ,    0.0330106...],
                       [ 731.        ,    0.0328284...],
                       [ 732.        ,    0.0326474...],
                       [ 733.        ,    0.0324677...],
                       [ 734.        ,    0.0322893...],
                       [ 735.        ,    0.0321120...],
                       [ 736.        ,    0.0319360...],
                       [ 737.        ,    0.0317611...],
                       [ 738.        ,    0.0315875...],
                       [ 739.        ,    0.0314151...],
                       [ 740.        ,    0.0312438...],
                       [ 741.        ,    0.0310737...],
                       [ 742.        ,    0.0309048...],
                       [ 743.        ,    0.0307370...],
                       [ 744.        ,    0.0305703...],
                       [ 745.        ,    0.0304048...],
                       [ 746.        ,    0.0302404...],
                       [ 747.        ,    0.0300771...],
                       [ 748.        ,    0.0299149...],
                       [ 749.        ,    0.0297538...],
                       [ 750.        ,    0.0295938...],
                       [ 751.        ,    0.0294349...],
                       [ 752.        ,    0.0292771...],
                       [ 753.        ,    0.0291203...],
                       [ 754.        ,    0.0289645...],
                       [ 755.        ,    0.0288098...],
                       [ 756.        ,    0.0286561...],
                       [ 757.        ,    0.0285035...],
                       [ 758.        ,    0.0283518...],
                       [ 759.        ,    0.0282012...],
                       [ 760.        ,    0.0280516...],
                       [ 761.        ,    0.0279030...],
                       [ 762.        ,    0.0277553...],
                       [ 763.        ,    0.0276086...],
                       [ 764.        ,    0.027463 ...],
                       [ 765.        ,    0.0273182...],
                       [ 766.        ,    0.0271744...],
                       [ 767.        ,    0.0270316...],
                       [ 768.        ,    0.0268897...],
                       [ 769.        ,    0.0267487...],
                       [ 770.        ,    0.0266087...],
                       [ 771.        ,    0.0264696...],
                       [ 772.        ,    0.0263314...],
                       [ 773.        ,    0.0261941...],
                       [ 774.        ,    0.0260576...],
                       [ 775.        ,    0.0259221...],
                       [ 776.        ,    0.0257875...],
                       [ 777.        ,    0.0256537...],
```

(continues on next page)

(continued from previous page)

```
                    [ 778.        ,    0.0255208...],
                    [ 779.        ,    0.0253888...],
                    [ 780.        ,    0.0252576...]],
                   interpolator=SpragueInterpolator,
                   interpolator_args={},
                   extrapolator=Extrapolator,
                   extrapolator_args={...})
```

colour.scattering_cross_section

colour.**scattering_cross_section**(*wavelength*, *CO2_concentration=300*, *temperature=288.15*, *avogadro_constant=6.02214179e+23*, *n_s=<function air_refraction_index_Bodhaine1999>*, *F_air=<function F_air_Bodhaine1999>*)

Returns the scattering cross section per molecule σ of dry air as function of wavelength λ in centimeters (cm) using given CO_2 concentration in parts per million (ppm) and temperature $T[K]$ in kelvin degrees following *Van de Hulst (1957)* method.

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **avogadro_constant** (numeric or array_like, optional) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6+3_p)/(6-7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns Scattering cross section per molecule σ of dry air.

Return type numeric or ndarray

Warning: Unlike most objects of colour.phenomena.rayleigh module, colour.scattering_cross_section() expects wavelength λ to be expressed in centimeters (cm).

References

[BWDS99], [Wik01d]

Examples

```
>>> scattering_cross_section(555 * 10e-8)
4.6613309...e-27
```

colour.phenomena

rayleigh_optical_depth(wavelength[, ...])	Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

colour.phenomena.rayleigh_optical_depth

colour.phenomena.**rayleigh_optical_depth**(*wavelength*, *CO2_concentration=300*, *temperature=288.15*, *pressure=101325*, *latitude=0*, *altitude=0*, *avogadro_constant=6.02214179e+23*, *n_s=<function air_refraction_index_Bodhaine1999>*, *F_air=<function F_air_Bodhaine1999>*)

Returns the *Rayleigh* optical depth $T_r(\lambda)$ as function of wavelength λ in centimeters (cm).

Parameters

- **wavelength** (numeric or array_like) – Wavelength λ in centimeters (cm).
- **CO2_concentration** (numeric or array_like, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric or array_like, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric or array_like) – Surface pressure P of the measurement site.
- **latitude** (numeric or array_like, optional) – Latitude of the site in degrees.
- **altitude** (numeric or array_like, optional) – Altitude of the site in meters.
- **avogadro_constant** (numeric or array_like, optional) – *Avogadro*'s number (molecules mol^{-1}).
- **n_s** (object) – Air refraction index n_s computation method.
- **F_air** (object) – $(6+3_p)/(6-7_p)$, the depolarisation term $F(air)$ or *King Factor* computation method.

Returns *Rayleigh* optical depth $T_r(\lambda)$.

Return type numeric or ndarray

Warning: Unlike most objects of colour.phenomena.rayleigh module, colour.phenomena.rayleigh_optical_depth() expects wavelength λ to be expressed in centimeters (cm).

References

[BWDS99], [Wik01d]

Examples

```
>>> rayleigh_optical_depth(555 * 10e-8)
0.1004070...
```

Plotting

- *Common*
- *Colorimetry*
- *Colour Vision Deficiency*
- *Colour Characterisation*
- *Corresponding Chromaticities*
- *CIE Chromaticity Diagrams*
- *Colour Models*
- *Colour Notation Systems*
- *Optical Phenomena*
- *Colour Quality*
- *Colour Temperature & Correlated Colour Temperature*
- *Colour Models Volume*
- *Geometry Plotting Utilities*
- *Automatic Colour Conversion Graph*

Common

`colour.plotting`

`colour_style`([use_style])	Returns *Colour* plotting style.
`colour_cycle`(**kwargs)	Returns a colour cycle iterator using given colour map.
`artist`(**kwargs)	Returns the current figure and its axes or creates a new one.
`camera`(**kwargs)	Sets the camera settings.
`render`(**kwargs)	Renders the current figure while adjusting various settings such as the bounding box, the title or background transparency.
`label_rectangles`(labels, rectangles[, ...])	Add labels above given rectangles.
`uniform_axes3d`(**kwargs)	Sets equal aspect ratio to given 3d axes.
`plot_single_colour_swatch`(colour_swatch, ...)	Plots given colour swatch.
`plot_multi_colour_swatches`(colour_swatches)	Plots given colours swatches.

Continued on next page

Table 205 – continued from previous page

plot_single_function(function[, samples, ...])	Plots given function.
plot_multi_functions(functions[, samples, ...])	Plots given functions.
plot_image(image[, text_parameters, ...])	Plots given image.

colour.plotting.colour_style

colour.plotting.**colour_style**(*use_style=True*)

Returns *Colour* plotting style.

Parameters **use_style** (bool, optional) – Whether to use the style and load it into *Matplotlib*.

Returns *Colour* style.

Return type dict

colour.plotting.colour_cycle

colour.plotting.**colour_cycle**(***kwargs*)

Returns a colour cycle iterator using given colour map.

Other Parameters

- **colour_cycle_map** (*unicode or LinearSegmentedColormap, optional*) – Matplotlib colourmap name.
- **colour_cycle_count** (*int, optional*) – Colours count to pick in the colourmap.

Returns Colour cycle iterator.

Return type cycle

colour.plotting.artist

colour.plotting.**artist**(***kwargs*)

Returns the current figure and its axes or creates a new one.

Other Parameters

- **axes** (*Axes, optional*) – Axes that will be passed through without creating a new figure.
- **uniform** (*unicode, optional*) – Whether to create the figure with an equal aspect ratio.

Returns Current figure and axes.

Return type tuple

colour.plotting.camera

colour.plotting.**camera**(***kwargs*)

Sets the camera settings.

Other Parameters

- **figure** (*Figure, optional*) – Figure to apply the render elements onto.
- **axes** (*Axes, optional*) – Axes to apply the render elements onto.
- **azimuth** (*numeric, optional*) – Camera azimuth.
- **camera_aspect** (*unicode, optional*) – Matplotlib axes aspect. Default is *equal*.
- **elevation** (*numeric, optional*) – Camera elevation.

Returns Current figure and axes.

Return type tuple

colour.plotting.render

colour.plotting.**render**(***kwargs*)

Renders the current figure while adjusting various settings such as the bounding box, the title or background transparency.

Other Parameters

- **figure** (*Figure, optional*) – Figure to apply the render elements onto.
- **axes** (*Axes, optional*) – Axes to apply the render elements onto.
- **filename** (*unicode, optional*) – Figure will be saved using given `filename` argument.
- **standalone** (*bool, optional*) – Whether to show the figure and call `plt.show()` definition.
- **aspect** (*unicode, optional*) – Matplotlib axes aspect.
- **axes_visible** (*bool, optional*) – Whether the axes are visible. Default is *True*.
- **bounding_box** (*array_like, optional*) – Array defining current axes limits such *bounding_box = (x min, x max, y min, y max)*.
- **tight_layout** (*bool, optional*) – Whether to invoke the `plt.tight_layout()` definition.
- **legend** (*bool, optional*) – Whether to display the legend. Default is *False*.
- **legend_columns** (*int, optional*) – Number of columns in the legend. Default is *1*.
- **transparent_background** (*bool, optional*) – Whether to turn off the background patch. Default is *False*.
- **title** (*unicode, optional*) – Figure title.
- **wrap_title** (*unicode, optional*) – Whether to wrap the figure title. Default is *True*.
- **x_label** (*unicode, optional*) – *X* axis label.
- **y_label** (*unicode, optional*) – *Y* axis label.
- **x_ticker** (*bool, optional*) – Whether to display the *X* axis ticker. Default is *True*.
- **y_ticker** (*bool, optional*) – Whether to display the *Y* axis ticker. Default is *True*.

Returns Current figure and axes.

Return type tuple

colour.plotting.label_rectangles

colour.plotting.**label_rectangles**(*labels*, *rectangles*, *rotation='vertical'*, *text_size=10*, *offset=None*, ***kwargs*)

Add labels above given rectangles.

Parameters

- **labels** (array_like) – Labels to display.
- **rectangles** (object) – Rectangles to used to set the labels value and position.
- **rotation** (unicode, optional) – **{'horizontal', 'vertical'}**, Labels orientation.
- **text_size** (numeric, optional) – Labels text size.
- **offset** (array_like, optional) – Labels offset as percentages of the largest rectangle dimensions.

Other Parameters

- **figure** (*Figure, optional*) – Figure to apply the render elements onto.
- **axes** (*Axes, optional*) – Axes to apply the render elements onto.

Returns Current figure and axes.

Return type tuple

colour.plotting.uniform_axes3d

colour.plotting.**uniform_axes3d**(***kwargs*)

Sets equal aspect ratio to given 3d axes.

Other Parameters

- **figure** (*Figure, optional*) – Figure to apply the render elements onto.
- **axes** (*Axes, optional*) – Axes to apply the render elements onto.

Returns Current figure and axes.

Return type tuple

colour.plotting.plot_single_colour_swatch

colour.plotting.**plot_single_colour_swatch**(*colour_swatch*, ***kwargs*)

Plots given colour swatch.

Parameters **colour_swatch** (ColourSwatch) – ColourSwatch.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_colour_swatches(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **width** (*numeric, optional*) – {colour.plotting.plot_multi_colour_swatches()}, Colour swatch width.
- **height** (*numeric, optional*) – {colour.plotting.plot_multi_colour_swatches()}, Colour swatch height.

- **spacing** (*numeric, optional*) – {colour.plotting.plot_multi_colour_swatches()}, Colour swatches spacing.
- **columns** (*int, optional*) – {colour.plotting.plot_multi_colour_swatches()}, Colour swatches columns count.
- **text_parameters** (*dict, optional*) – {colour.plotting.plot_multi_colour_swatches()}, Parameters for the `plt.text()` definition, `offset` can be set to define the text offset.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = ColourSwatch(RGB=(0.45620519, 0.03081071, 0.04091952))
>>> plot_single_colour_swatch(RGB)
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

colour.plotting.plot_multi_colour_swatches

colour.plotting.**plot_multi_colour_swatches**(*colour_swatches*, *width=1*, *height=1*, *spacing=0*, *columns=None*, *text_parameters=None*, *background_colour=(1.0, 1.0, 1.0)*, *compare_swatches=None*, ***kwargs*)

Plots given colours swatches.

Parameters

- **colour_swatches** (list) – Colour swatch sequence.
- **width** (numeric, optional) – Colour swatch width.
- **height** (numeric, optional) – Colour swatch height.
- **spacing** (numeric, optional) – Colour swatches spacing.
- **columns** (int, optional) – Colour swatches columns count, defaults to the colour swatch count or half of it if comparing.

- **text_parameters** (dict, optional) – Parameters for the plt.text() definition, visible can be set to make the text visible, offset can be set to define the text offset.
- **background_colour** (array_like or unicode, optional) – Background colour.
- **compare_swatches** (unicode, optional) – **{None, 'Stacked', 'Diagonal'}**, Whether to compare the swatches, in which case the colour swatch count must be an even number with alternating reference colour swatches and test colour swatches. *Stacked* will draw the test colour swatch in the center of the reference colour swatch, *Diagonal* will draw the reference colour swatch in the upper left diagonal area and the test colour swatch in the bottom right diagonal area.

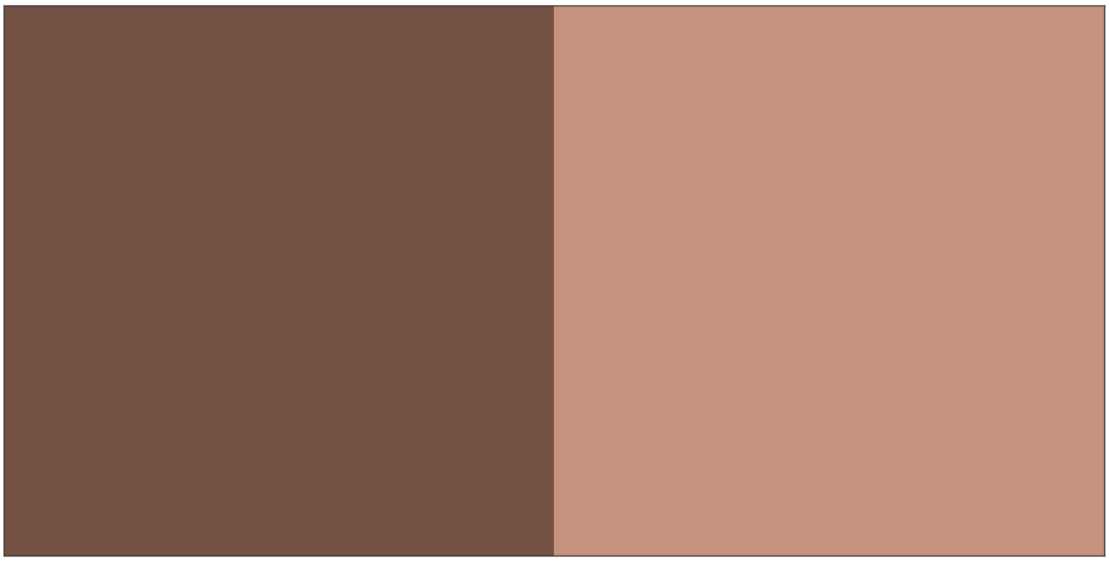
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB_1 = ColourSwatch(RGB=(0.45293517, 0.31732158, 0.26414773))
>>> RGB_2 = ColourSwatch(RGB=(0.77875824, 0.57726450, 0.50453169))
>>> plot_multi_colour_swatches([RGB_1, RGB_2])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_function

colour.plotting.**plot_single_function**(*function*, *samples=None*, *log_x=None*, *log_y=None*, ***kwargs*)

Plots given function.

Parameters

- **function** (callable, optional) – Function to plot.
- **samples** (array_like, optional,) – Samples to evaluate the functions with.
- **log_x** (int, optional) – Log base to use for the *x* axis scale, if *None*, the *x* axis scale will be linear.
- **log_y** (int, optional) – Log base to use for the *y* axis scale, if *None*, the *y* axis scale will be linear.

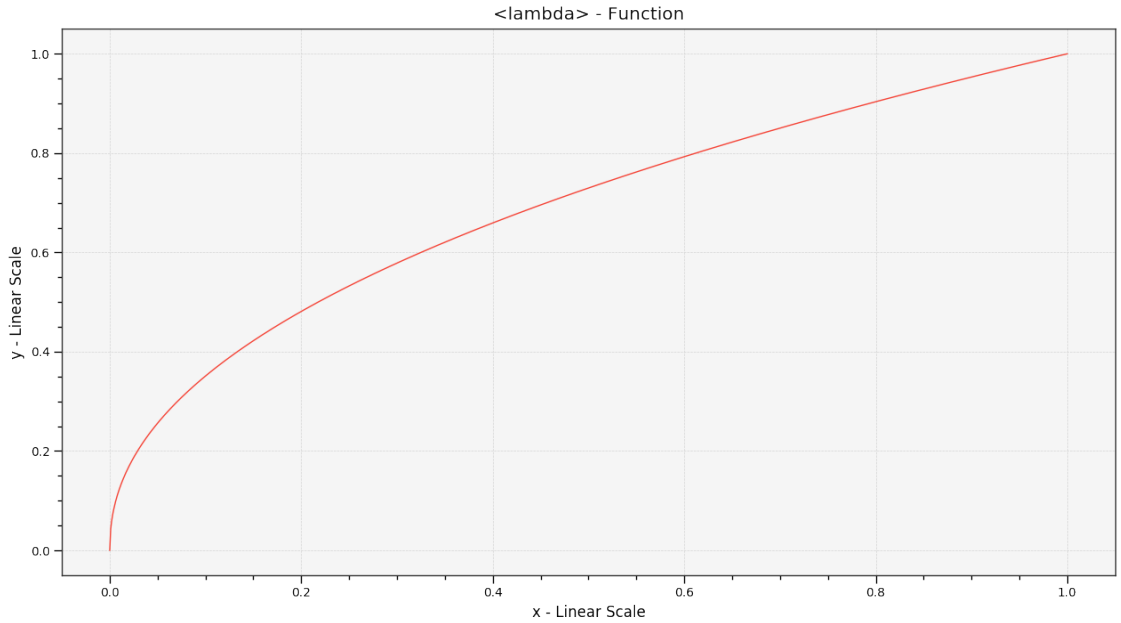
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour.models import gamma_function
>>> plot_single_function(partial(gamma_function, exponent=1 / 2.2))
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_functions

colour.plotting.**plot_multi_functions**(*functions*, *samples=None*, *log_x=None*, *log_y=None*, ***kwargs*)

Plots given functions.

Parameters

- **functions** (dict) – Functions to plot.
- **samples** (array_like, optional,) – Samples to evaluate the functions with.
- **log_x** (int, optional) – Log base to use for the *x* axis scale, if *None*, the *x* axis scale will be linear.
- **log_y** (int, optional) – Log base to use for the *y* axis scale, if *None*, the *y* axis scale will be linear.

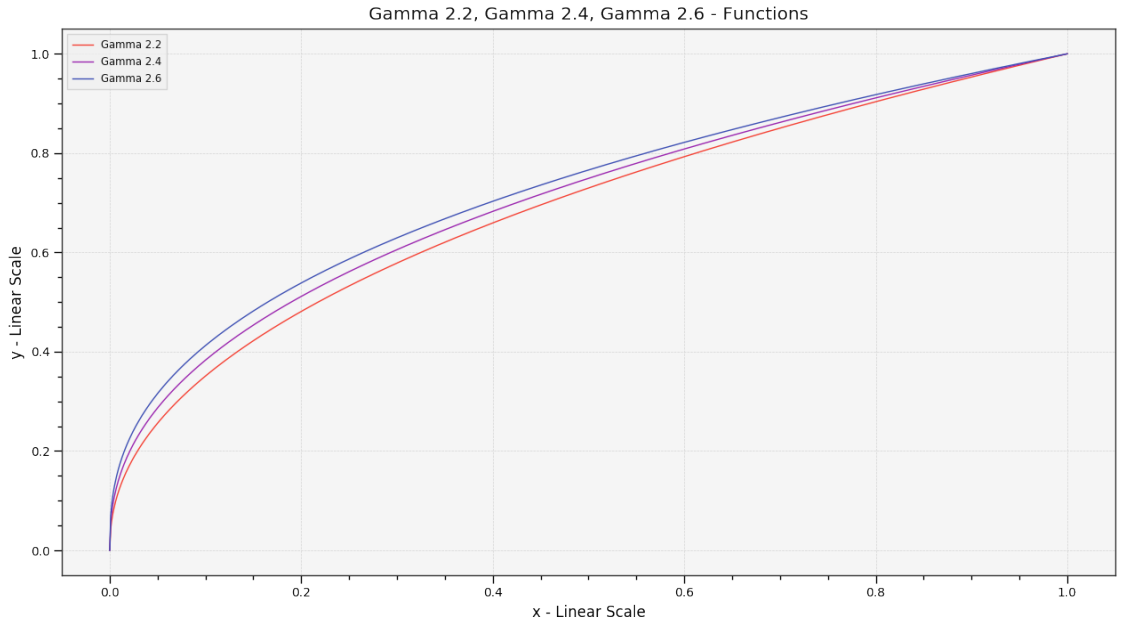
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> functions = {
...     'Gamma 2.2' : lambda x: x ** (1 / 2.2),
...     'Gamma 2.4' : lambda x: x ** (1 / 2.4),
...     'Gamma 2.6' : lambda x: x ** (1 / 2.6),
... }
>>> plot_multi_functions(functions)
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_image

colour.plotting.**plot_image**(*image*, *text_parameters=None*, *interpolation='nearest'*, *colour_map=<matplotlib.colors.LinearSegmentedColormap object>*, ***kwargs*)

Plots given image.

Parameters

- **image** (array_like) – Image to plot.
- **text_parameters** (dict, optional) – Parameters for the plt.text() definition, offset can be set to define the text offset.
- **interpolation** (unicode, optional) – **{'nearest', None, 'none', 'bilinear', 'bicubic', 'spline16', 'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric', 'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos'}** Image display interpolation.
- **colour_map** (unicode, optional) – Colour map used to display single channel images.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> import os
>>> import colour
>>> from colour import read_image
>>> path = os.path.join(
...     colour.__path__[0], '..', 'docs', '_static', 'Logo_Medium_001.png')
>>> plot_image(read_image(path))
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

Colorimetry

colour.plotting

plot_single_sd(sd[, cmfs, ...])	Plots given spectral distribution.
plot_multi_sds(sds[, cmfs, use_sds_colours, ...])	Plots given spectral distributions.
plot_single_cmfs([cmfs])	Plots given colour matching functions.
plot_multi_cmfs([cmfs])	Plots given colour matching functions.
plot_single_illuminant_sd([illuminant, cmfs])	Plots given single illuminant spectral distribution.
plot_multi_illuminant_sds([illuminants])	Plots given illuminants spectral distributions.
plot_visible_spectrum([cmfs, ...])	Plots the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.
plot_single_lightness_function([function])	Plots given *Lightness* function.
plot_multi_lightness_functions([functions])	Plots given *Lightness* functions.
plot_single_luminance_function([function])	Plots given *Luminance* function.
plot_multi_luminance_functions([functions])	Plots given *Luminance* functions.
plot_blackbody_spectral_radiance([...])	Plots given blackbody spectral radiance.
plot_blackbody_colours([shape, cmfs])	Plots blackbody colours.

colour.plotting.plot_single_sd

colour.plotting.**plot_single_sd**(*sd*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *out_of_gamut_clipping=True*, *modulate_colours_with_sd_amplitude=False*, *equalize_sd_amplitude=False*, ***kwargs*)

Plots given spectral distribution.

Parameters

- **sd** (SpectralDistribution) – Spectral distribution to plot.
- **out_of_gamut_clipping** (bool, optional) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.
- **modulate_colours_with_sd_amplitude** (bool, optional) – Whether to modulate the colours with the spectral distribution amplitude.
- **equalize_sd_amplitude** (bool, optional) – Whether to equalize the spectral distribution amplitude. Equalization occurs after the colours modulation thus setting both arguments to *True* will generate a spectrum strip where each wavelength colour is modulated by the spectral distribution amplitude. The usual 5% margin above the spectral distribution is also omitted.
- **cmfs** (unicode) – Standard observer colour matching functions used for spectrum creation.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

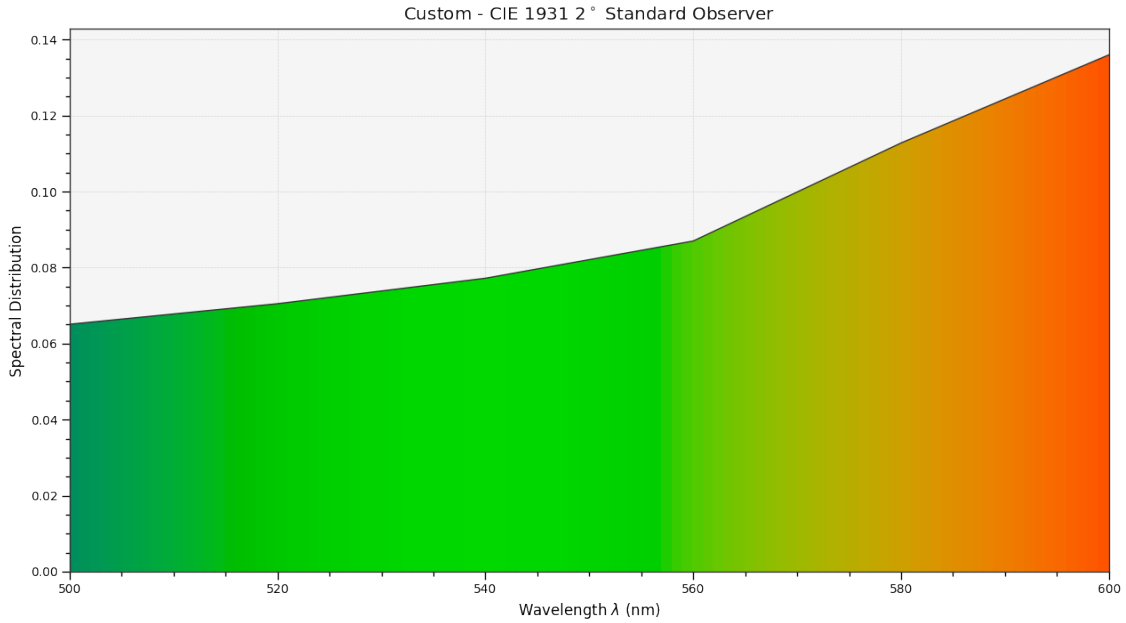
Return type tuple

References

[Spi15]

Examples

```
>>> from colour import SpectralDistribution
>>> data = {
...     500: 0.0651,
...     520: 0.0705,
...     540: 0.0772,
...     560: 0.0870,
...     580: 0.1128,
...     600: 0.1360
... }
>>> sd = SpectralDistribution(data, name='Custom')
>>> plot_single_sd(sd)
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_sds

colour.plotting.**plot_multi_sds**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *use_sds_colours=False*, *normalise_sds_colours=False*, ***kwargs*)

Plots given spectral distributions.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectrum creation.

- **use_sds_colours** (bool, optional) – Whether to use spectral distributions colours.
- **normalise_sds_colours** (bool) – Whether to normalise spectral distributions colours.

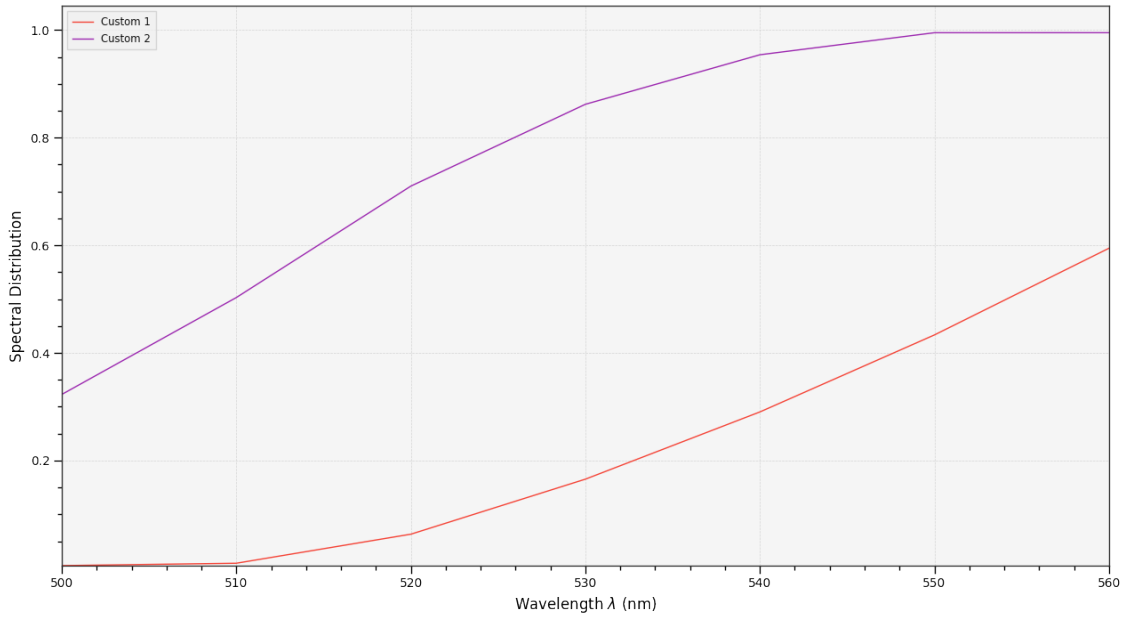
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import SpectralDistribution
>>> data_1 = {
...     500: 0.004900,
...     510: 0.009300,
...     520: 0.063270,
...     530: 0.165500,
...     540: 0.290400,
...     550: 0.433450,
...     560: 0.594500
... }
>>> data_2 = {
...     500: 0.323000,
...     510: 0.503000,
...     520: 0.710000,
...     530: 0.862000,
...     540: 0.954000,
...     550: 0.994950,
...     560: 0.995000
... }
>>> sd_1 = SpectralDistribution(data_1, name='Custom 1')
>>> sd_2 = SpectralDistribution(data_2, name='Custom 2')
>>> plot_multi_sds([sd_1, sd_2])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_cmfs

colour.plotting.**plot_single_cmfs**(*cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)
Plots given colour matching functions.

Parameters **cmfs** (unicode, optional) – Colour matching functions to plot.

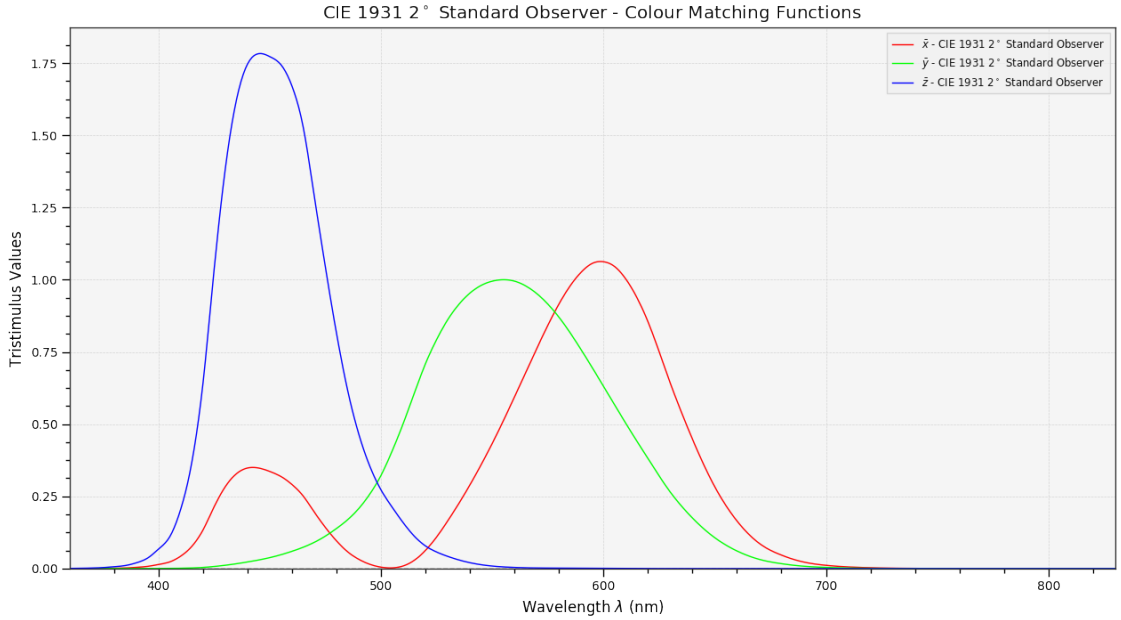
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_cmfs(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_cmfs('CIE 1931 2 Degree Standard Observer')
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_cmfs

colour.plotting.**plot_multi_cmfs**(*cmfs=None*, ***kwargs*)
Plots given colour matching functions.

Parameters **cmfs** (array_like, optional) – Colour matching functions to plot.

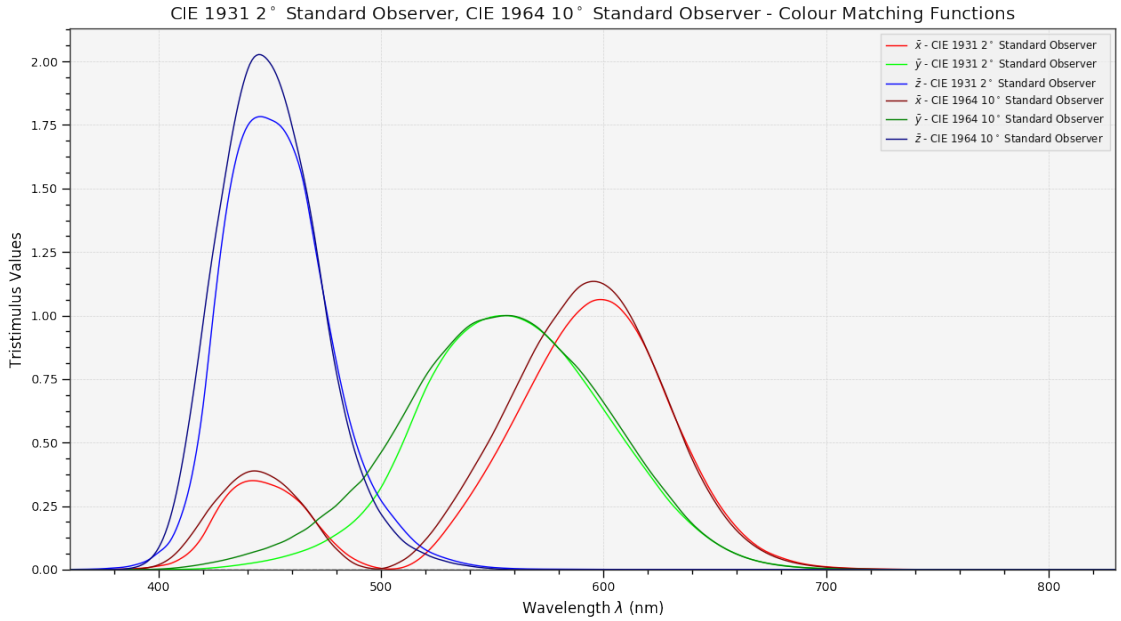
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> cmfs = ('CIE 1931 2 Degree Standard Observer',
...         'CIE 1964 10 Degree Standard Observer')
>>> plot_multi_cmfs(cmfs)
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_illuminant_sd

colour.plotting.**plot_single_illuminant_sd**(*illuminant='A'*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots given single illuminant spectral distribution.

Parameters

- **illuminant** (unicode, optional) – Factory illuminant to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions to plot.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **out_of_gamut_clipping** (*bool, optional*) – {colour.plotting.plot_single_sd()}, Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Returns Current figure and axes.

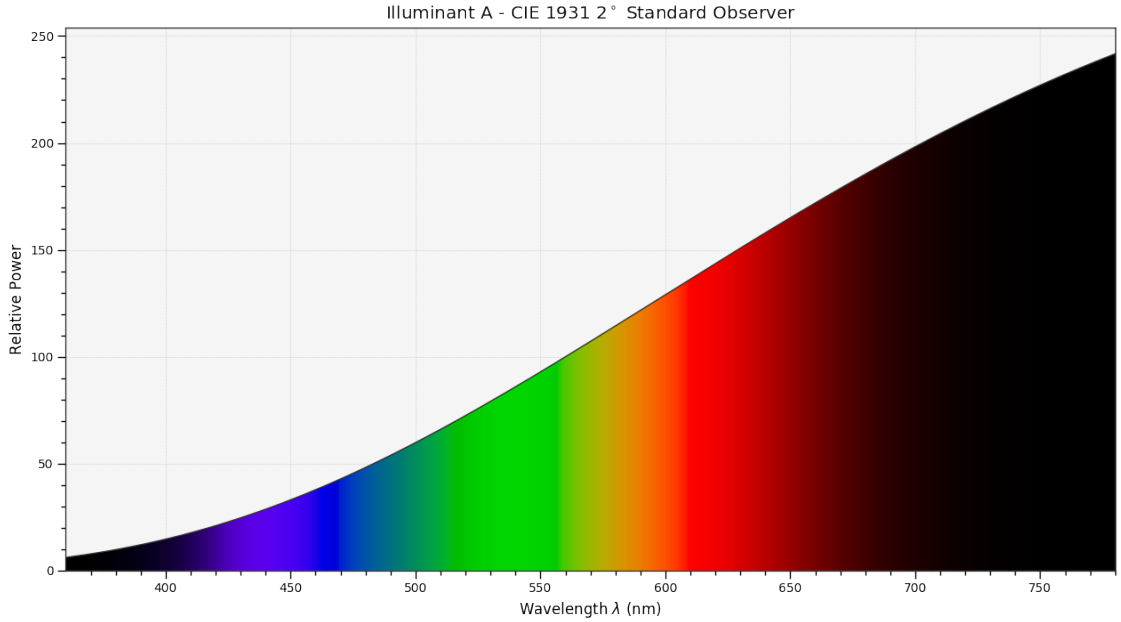
Return type tuple

References

[Spi15]

Examples

```
>>> plot_single_illuminant_sd('A')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_illuminant_sds

colour.plotting.**plot_multi_illuminant_sds**(*illuminants=None*, ***kwargs*)

Plots given illuminants spectral distributions.

Parameters **illuminants** (array_like, optional) – Factory illuminants to plot.

Other Parameters

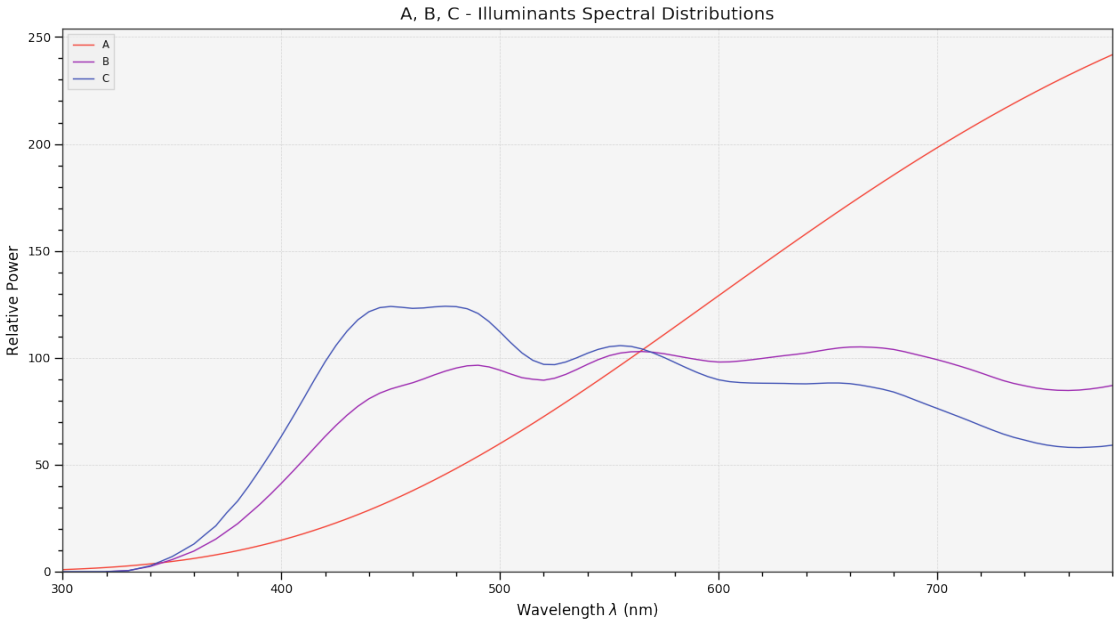
- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_sds(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **use_sds_colours** (*bool, optional*) – {colour.plotting.plot_multi_sds()} Whether to use spectral distributions colours.
- **normalise_sds_colours** (*bool*) – {colour.plotting.plot_multi_sds()} Whether to normalise spectral distributions colours.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_illuminant_sds(['A', 'B', 'C'])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_visible_spectrum

colour.plotting.**plot_visible_spectrum**(*cmfs='CIE 1931 2 Degree Standard Observer'*, *out_of_gamut_clipping=True*, ***kwargs*)

Plots the visible colours spectrum using given standard observer *CIE XYZ* colour matching functions.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectrum creation.
- **out_of_gamut_clipping** (bool, optional) – Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

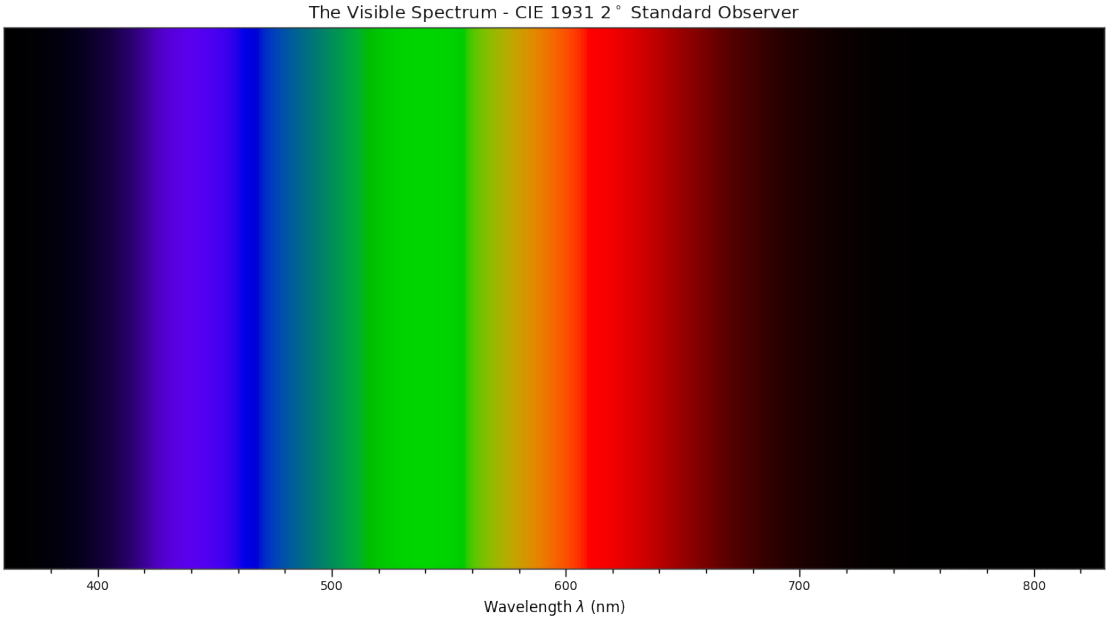
Return type tuple

References

[Spi15]

Examples

```
>>> plot_visible_spectrum()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_lightness_function

colour.plotting.**plot_single_lightness_function**(*function='CIE 1976'*, ***kwargs*)

Plots given *Lightness* function.

Parameters **function** (unicode, optional) – *Lightness* function to plot.

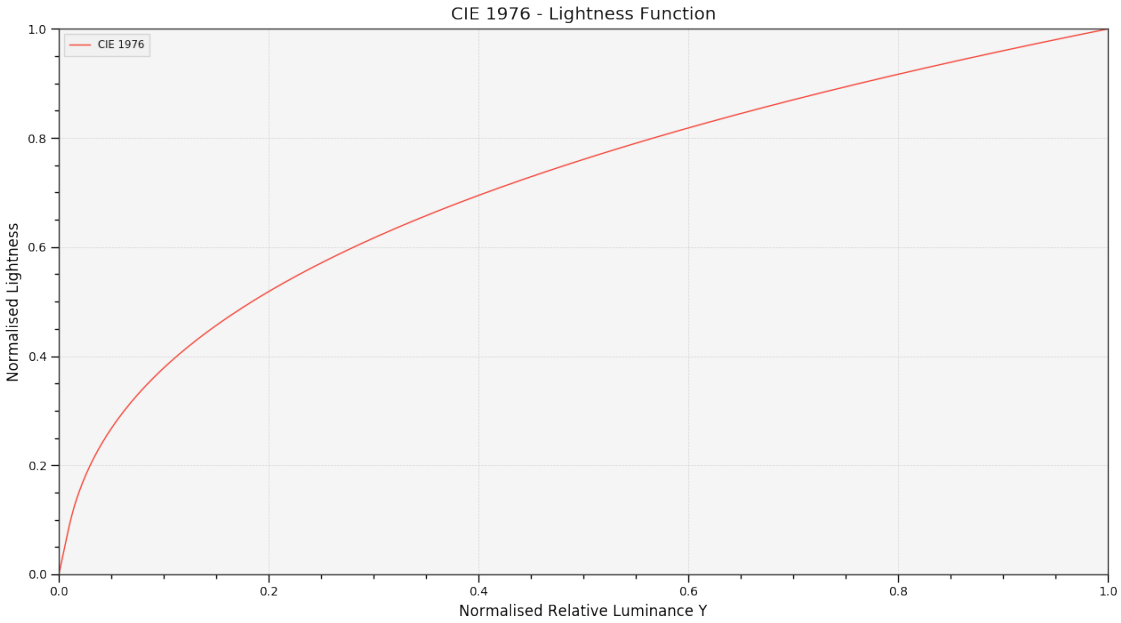
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_lightness_function('CIE 1976')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_lightness_functions

colour.plotting.**plot_multi_lightness_functions**(*functions=None*, ***kwargs*)

Plots given *Lightness* functions.

Parameters **functions** (array_like, optional) – *Lightness* functions to plot.

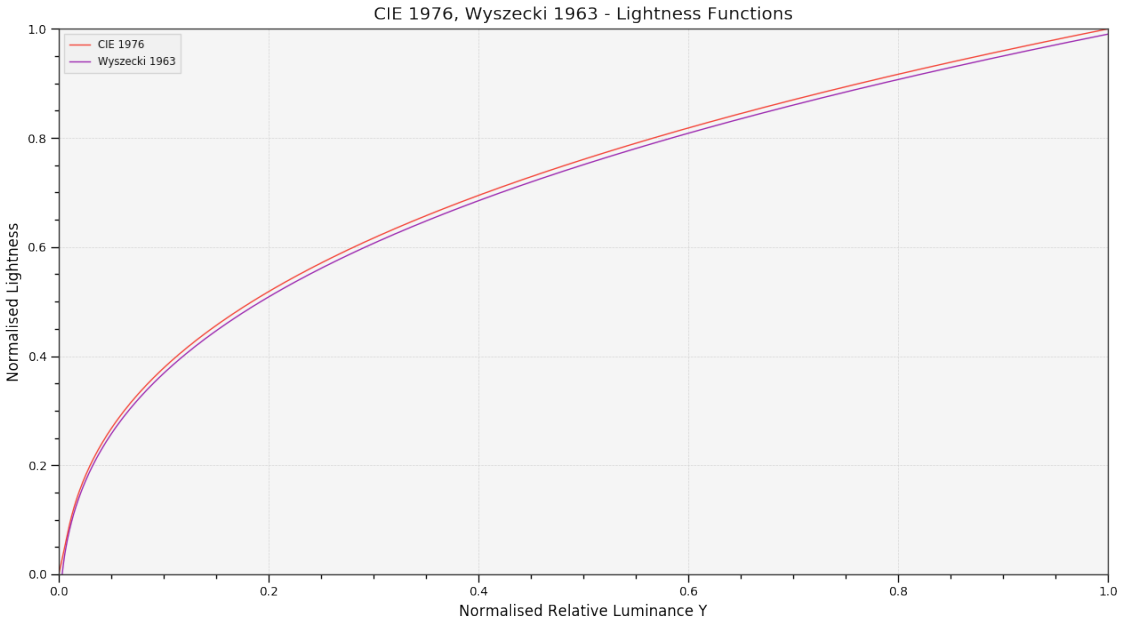
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_lightness_functions(['CIE 1976', 'Wyszecki 1963'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_luminance_function

colour.plotting.**plot_single_luminance_function**(*function='CIE 1976'*, ***kwargs*)

Plots given *Luminance* function.

Parameters **function** (unicode, optional) – *Luminance* function to plot.

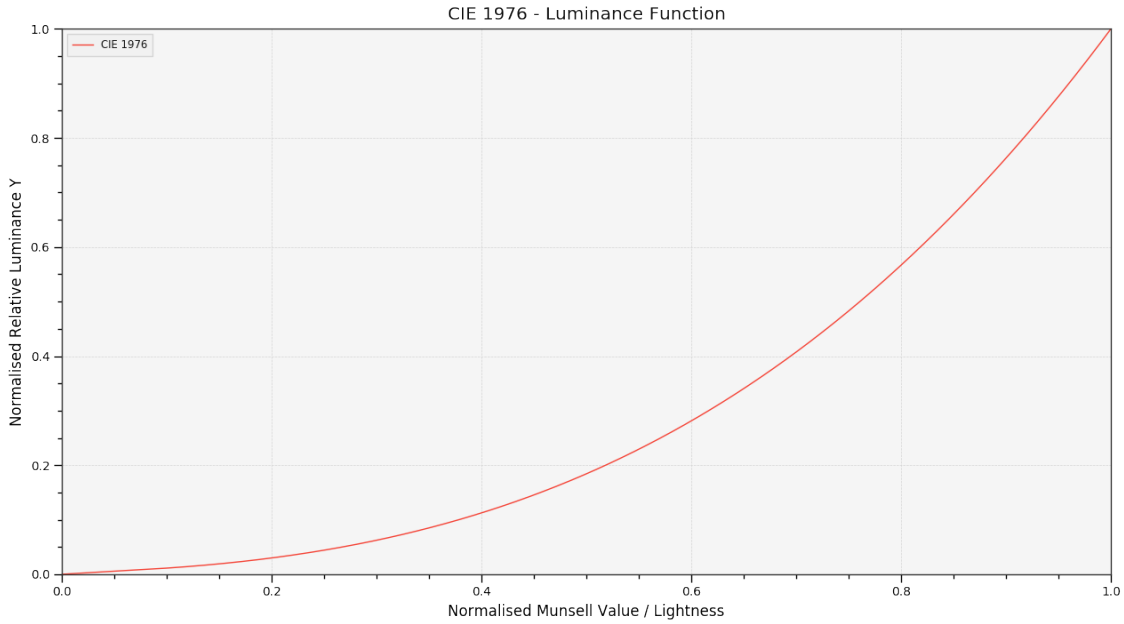
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_luminance_function('CIE 1976')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_luminance_functions

colour.plotting.**plot_multi_luminance_functions**(*functions=None*, ***kwargs*)

Plots given *Luminance* functions.

Parameters **functions** (array_like, optional) – *Luminance* functions to plot.

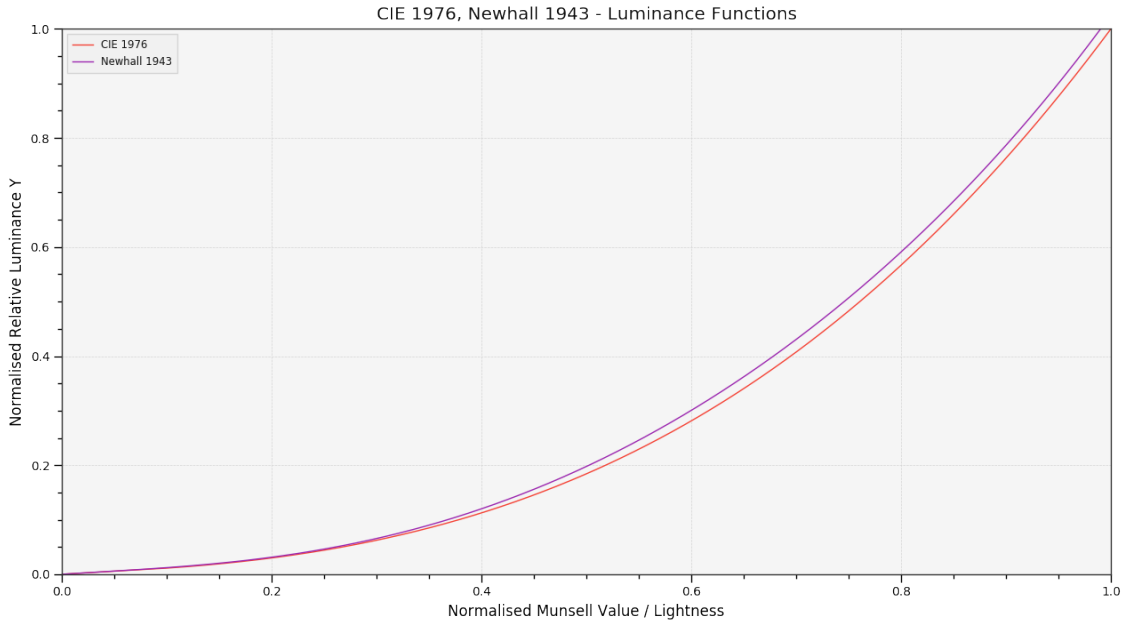
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_luminance_functions(['CIE 1976', 'Newhall 1943'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_blackbody_spectral_radiance

colour.plotting.**plot_blackbody_spectral_radiance**(*temperature=3500*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *blackbody='VY Canis Major'*, ***kwargs*)

Plots given blackbody spectral radiance.

Parameters

- **temperature** (numeric, optional) – Blackbody temperature.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.
- **blackbody** (unicode, optional) – Blackbody name.

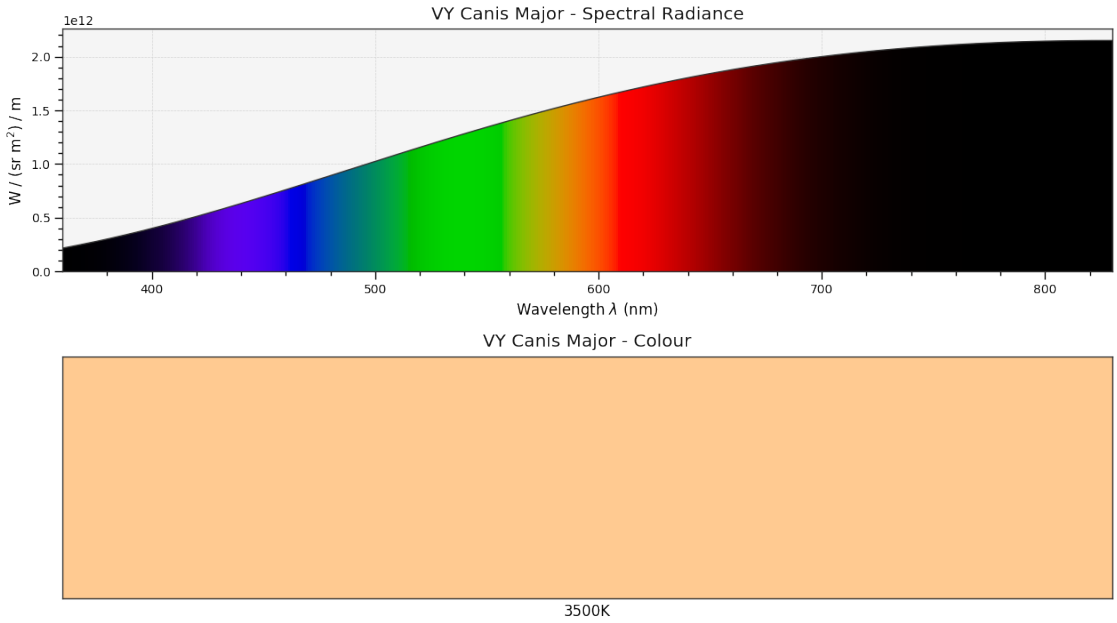
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_blackbody_spectral_radiance(3500, blackbody='VY Canis Major')
...
(<Figure size ... with 2 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_blackbody_colours

`colour.plotting.`**`plot_blackbody_colours`**(*shape=SpectralShape(150, 12500, 50)*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots blackbody colours.

Parameters

- **shape** (`SpectralShape`, `optional`) – Spectral shape to use as plot boundaries.
- **cmfs** (`unicode`, `optional`) – Standard observer colour matching functions.

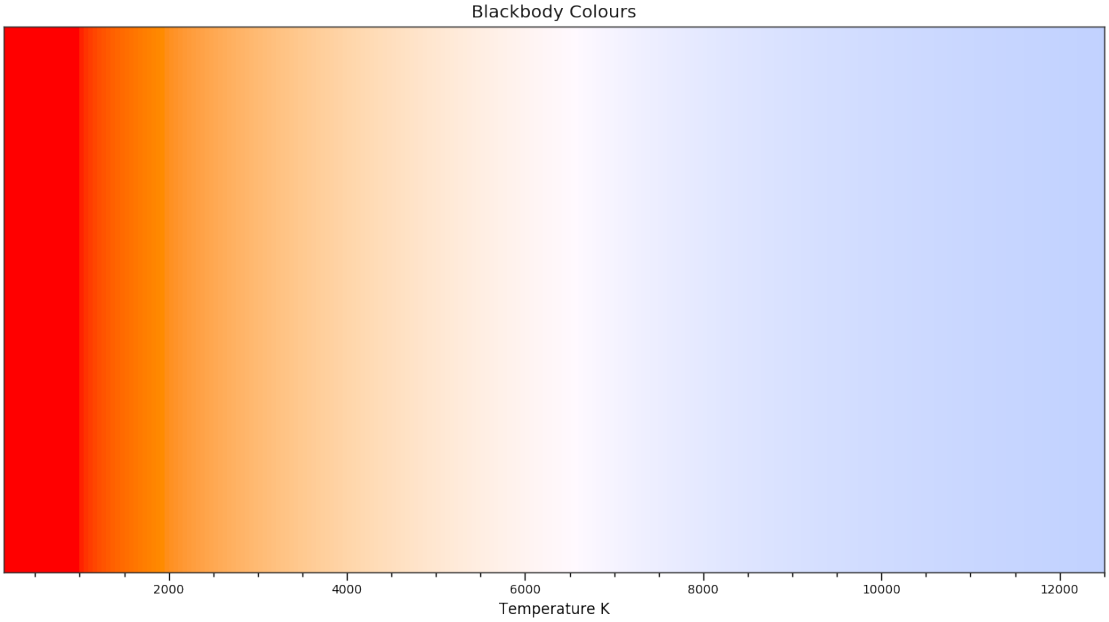
Other Parameters ****kwargs** (*dict*, *optional*) – {`colour.plotting.artist()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_blackbody_colours(SpectralShape(150, 12500, 50))
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Vision Deficiency

colour.plotting

plot_cvd_simulation_Machado2009(RGB[, ...])	Performs colour vision deficiency simulation on given *RGB* colourspace array using *Machado et al.(2009)* model..

colour.plotting.plot_cvd_simulation_Machado2009

colour.plotting.**plot_cvd_simulation_Machado2009**(*RGB*, *deficiency='Protanomaly'*, *severity=0.5*, *M_a=None*, ***kwargs*)

Performs colour vision deficiency simulation on given *RGB* colourspace array using *Machado et al. (2009)* model.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **deficiency** (unicode, optional) – {'Protanomaly', 'Deuteranomaly', 'Tritanomaly'} Colour blindness / vision deficiency type.
- **severity** (numeric, optional) – Severity of the colour vision deficiency in domain [0, 1].
- **M_a** (array_like, optional) – Anomalous trichromacy matrix to use instead of Machado (2010) pre-computed matrix.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_image(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Notes

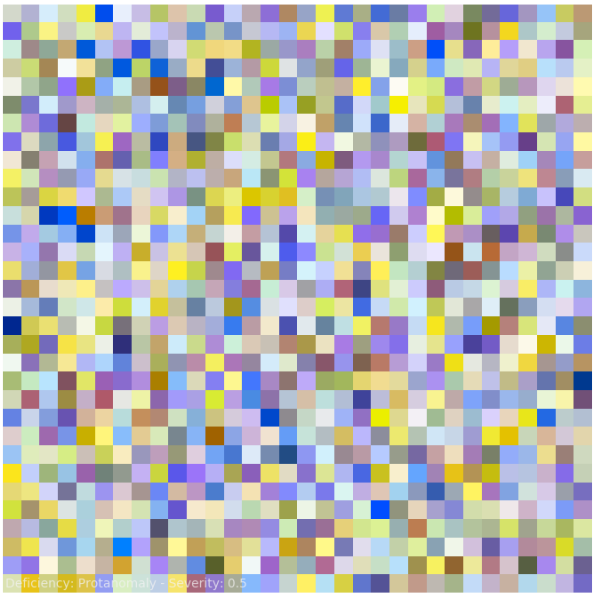
- Input *RGB* array is expected to be linearly encoded.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> import numpy as np
>>> RGB = np.random.rand(32, 32, 3)
>>> plot_cvd_simulation_Machado2009(RGB)
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Characterisation

`colour.plotting`

`plot_single_colour_checker`([colour_checker])	Plots given colour checker.
`plot_multi_colour_checkers`([colour_checkers])	Plots and compares given colour checkers.

colour.plotting.plot_single_colour_checker

`colour.plotting.`**`plot_single_colour_checker`**(*colour_checker='ColorChecker 2005', **kwargs*)

Plots given colour checker.

Parameters **colour_checker** (unicode, optional) – Color checker name.

Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_colour_swatches()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_colour_checker('ColorChecker 2005')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

colour.plotting.plot_multi_colour_checkers

colour.plotting.**plot_multi_colour_checkers**(*colour_checkers=None*, ***kwargs*)

Plots and compares given colour checkers.

Parameters **colour_checkers** (array_like, optional) – Color checker names, must be less than or equal to 2 names.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_colour_swatches(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_colour_checkers(['ColorChecker 1976', 'ColorChecker 2005'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

Corresponding Chromaticities

colour.plotting

plot_corresponding_chromaticities_prediction([...])	Plots given chromatic adaptation model corresponding chromaticities prediction.

colour.plotting.plot_corresponding_chromaticities_prediction

colour.plotting.**plot_corresponding_chromaticities_prediction**(*experiment=1*, *model='Von Kries'*, *transform='CAT02'*, ***kwargs*)

Plots given chromatic adaptation model corresponding chromaticities prediction.

Parameters

- **experiment** (integer or CorrespondingColourDataset, optional) – {1, 2, 3, 4, 6, 8, 9, 11, 12} *Breneman (1987)* experiment number or colour.CorrespondingColourDataset class instance.
- **model** (unicode, optional) – Corresponding chromaticities prediction model name.
- **transform** (unicode, optional) – Transformation to use with *Von Kries* chromatic adaptation model.

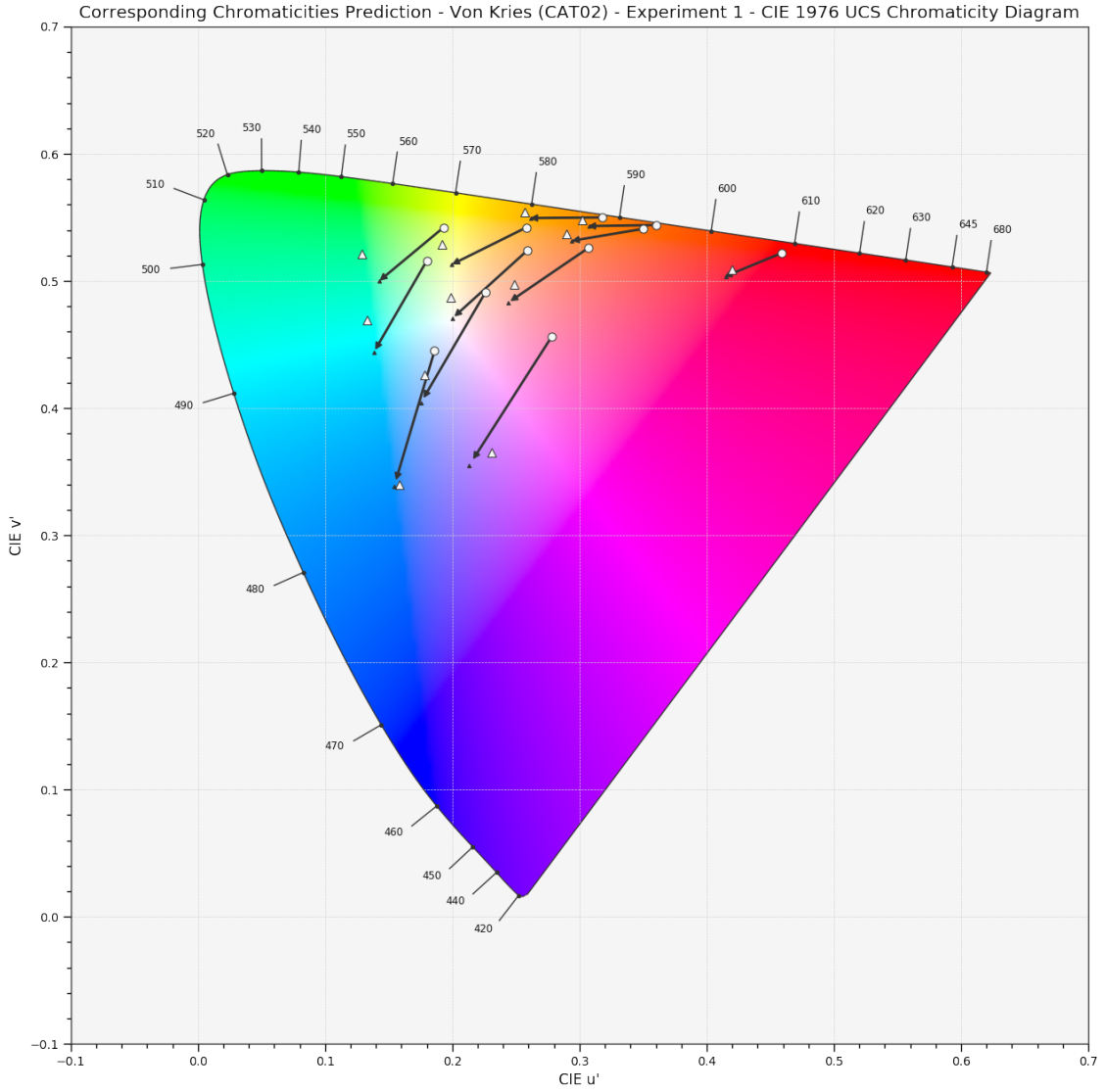
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_corresponding_chromaticities_prediction(1, 'Von Kries', 'CAT02')
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



CIE Chromaticity Diagrams

`colour.plotting`

plot_chromaticity_diagram_CIE1931([cmfs, ...])	Plots the *CIE 1931 Chromaticity Diagram*.
plot_chromaticity_diagram_CIE1960UCS([cmfs, ...])	Plots the *CIE 1960 UCS Chromaticity Diagram*.
plot_chromaticity_diagram_CIE1976UCS([cmfs, ...])	Plots the *CIE 1976 UCS Chromaticity Diagram*.

Continued on next page

Table 210 – continued from previous page

plot_sds_in_chromaticity_diagram_CIE1931(sds)	Plots given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1960UCS(sds)	Plots given spectral distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.
plot_sds_in_chromaticity_diagram_CIE1976UCS(sds)	Plots given spectral distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

colour.plotting.plot_chromaticity_diagram_CIE1931

colour.plotting.**plot_chromaticity_diagram_CIE1931**(*cmfs='CIE 1931 2 Degree Standard Observer'*, *show_diagram_colours=True*, *show_spectral_locus=True*, ***kwargs*)

Plots the *CIE 1931 Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

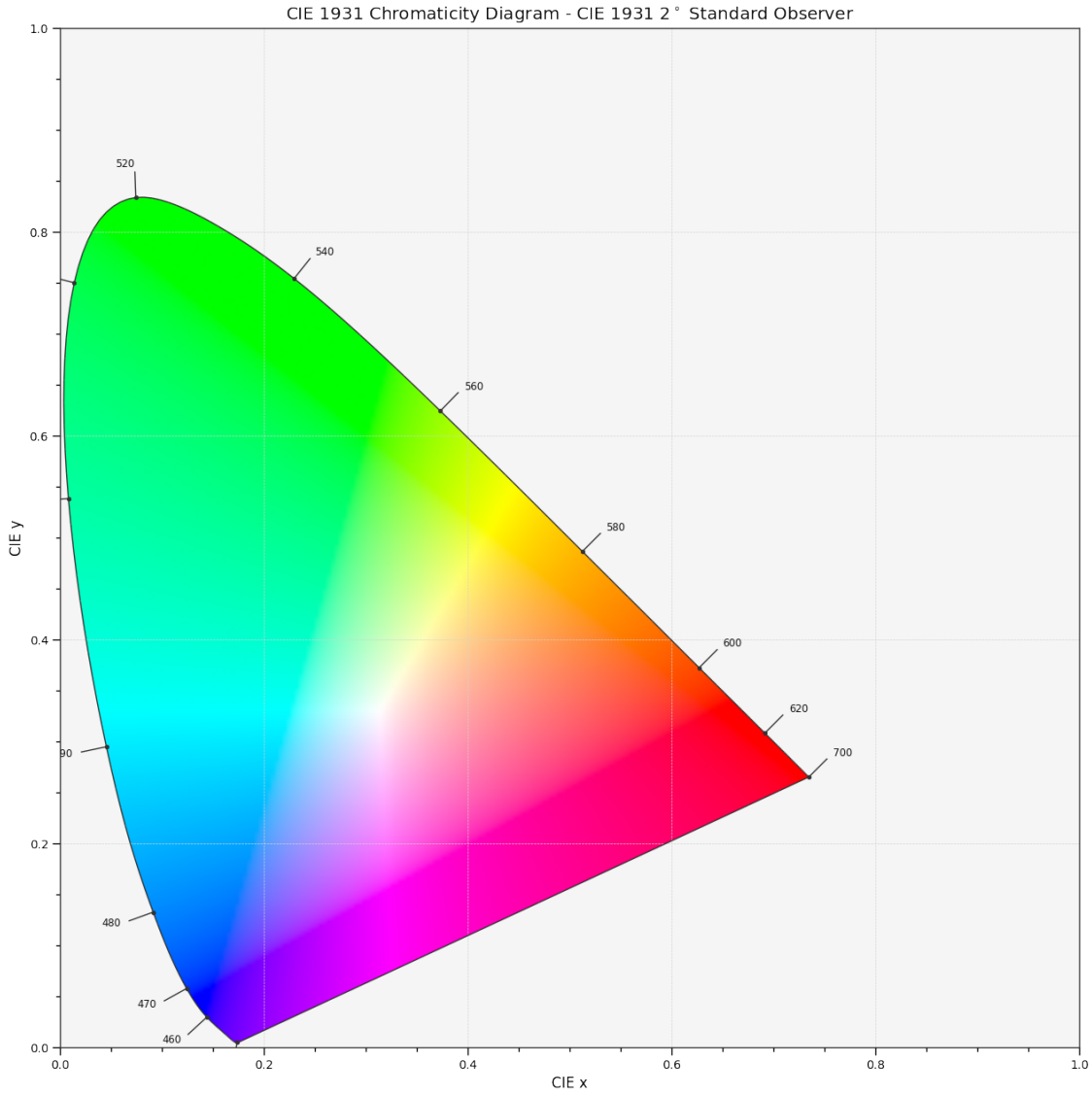
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1931()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_chromaticity_diagram_CIE1960UCS**(*cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, show_spectral_locus=True, **kwargs*)

Plots the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

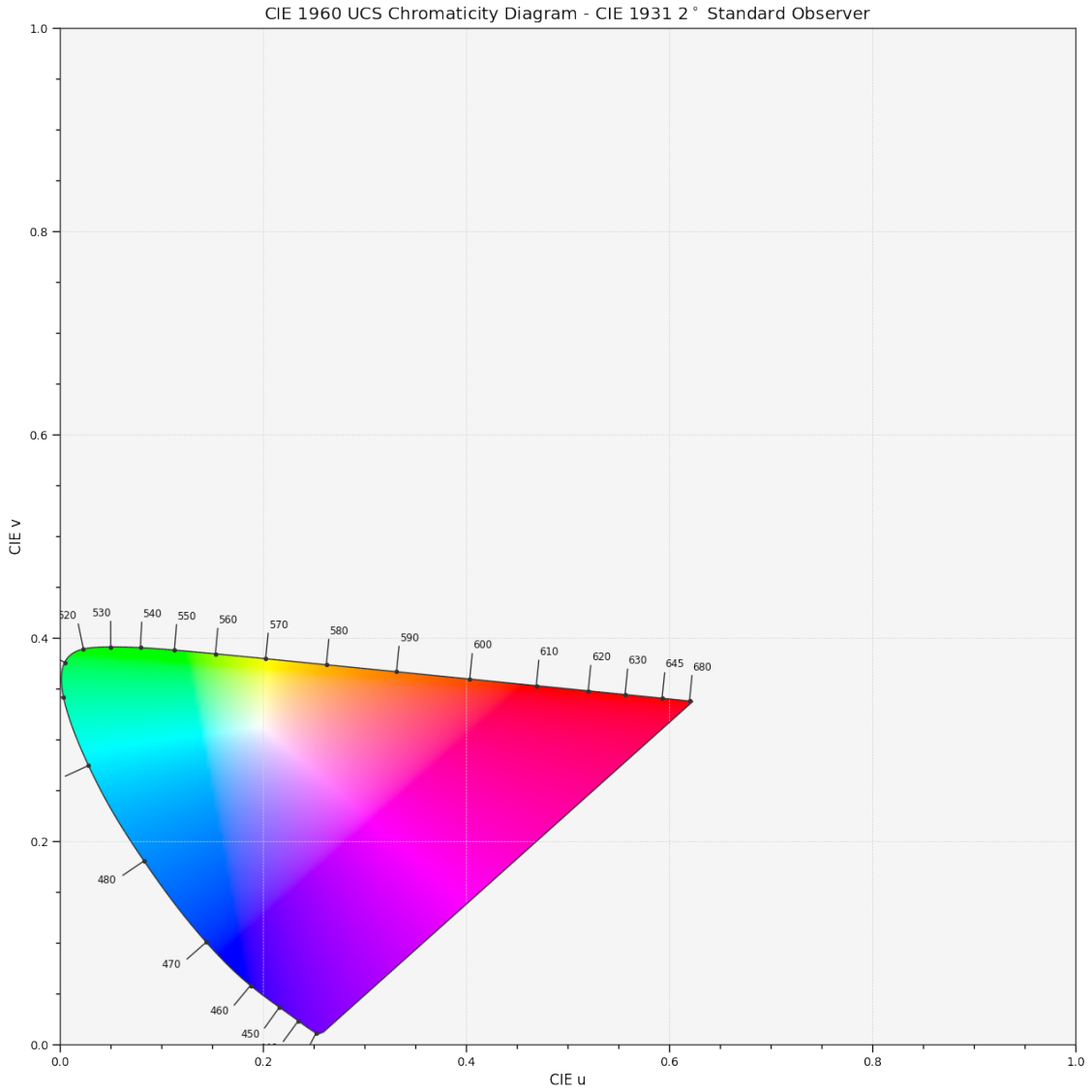
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1960UCS()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_chromaticity_diagram_CIE1976UCS**(*cmfs='CIE 1931 2 Degree Standard Observer', show_diagram_colours=True, show_spectral_locus=True, **kwargs*)

Plots the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.

- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.

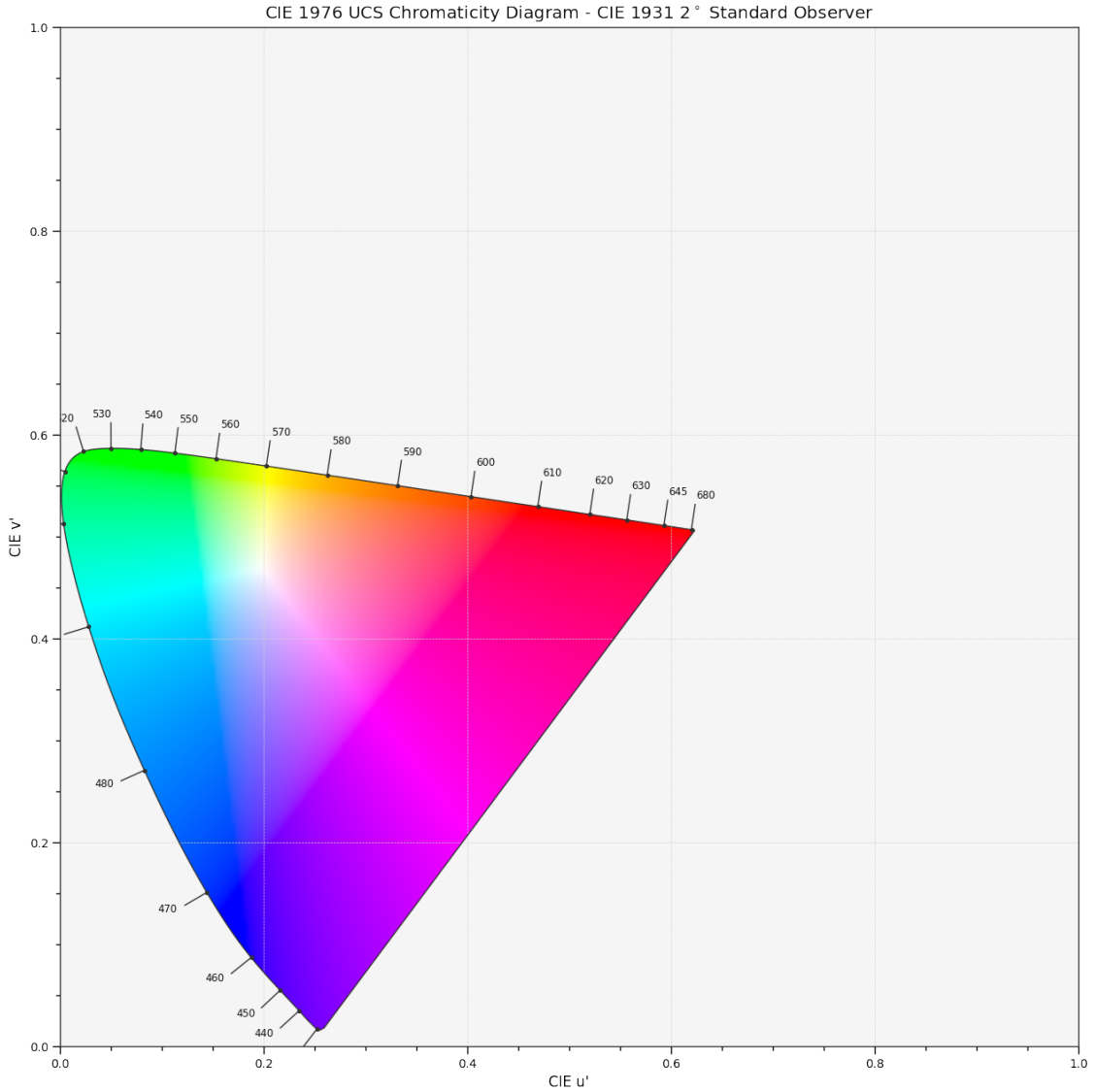
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_CIE1976UCS()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1931**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate_parameters=None*, *chromaticity_diagram_callable_CIE1931=<function plot_chromaticity_diagram_CIE1931>*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *CIE 1931 Chromaticity Diagram*.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if annotate is set to *True*. annotate_parameters can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

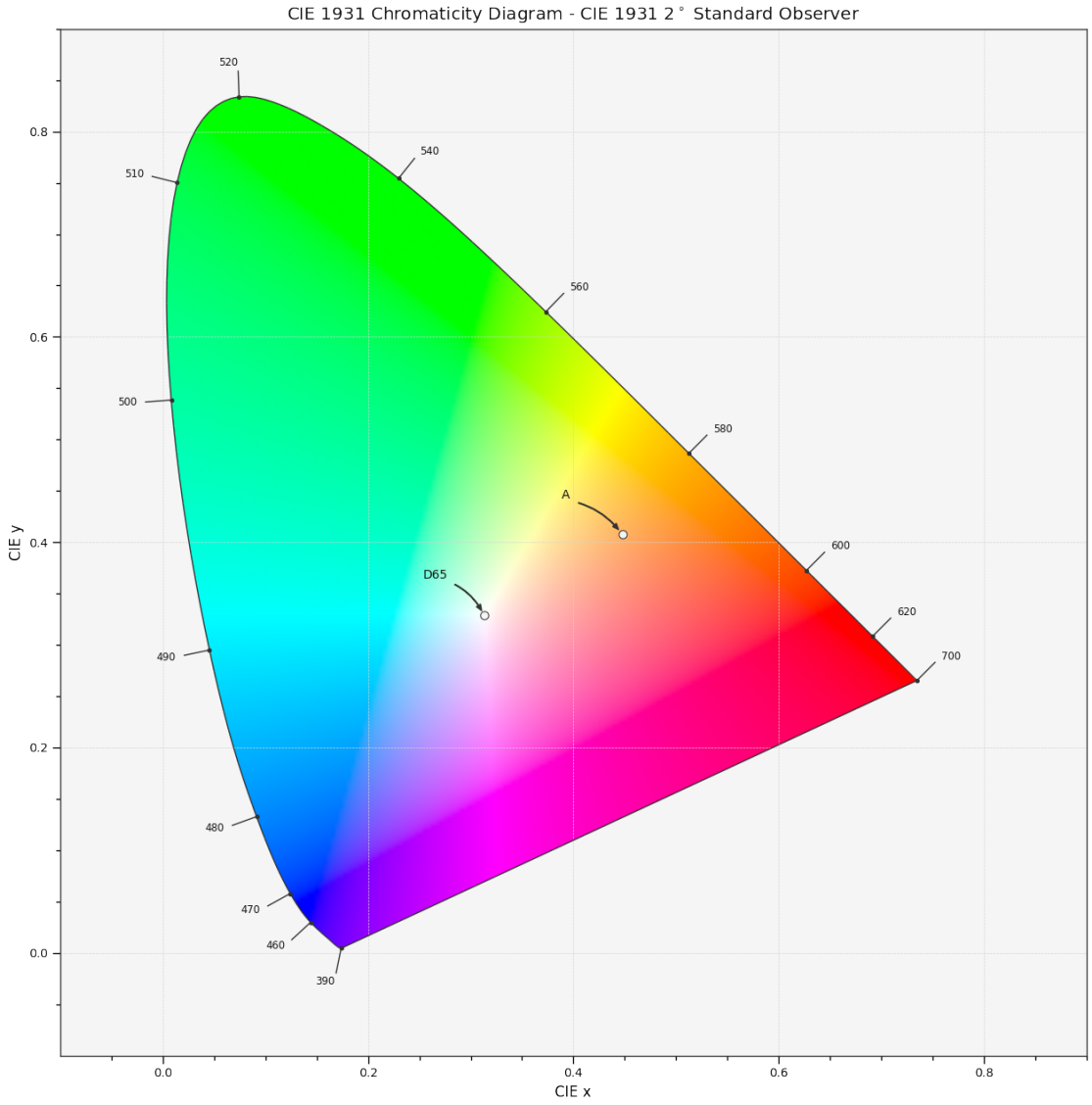
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1931([A, D65])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1960UCS**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate_parameters=None*, *chromaticity_diagram_callable_CIE1960UCS=<function plot_chromaticity_diagram_CIE1960UCS>*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.

- **annotate_parameters** (dict or array_like, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if `annotate` is set to *True*. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

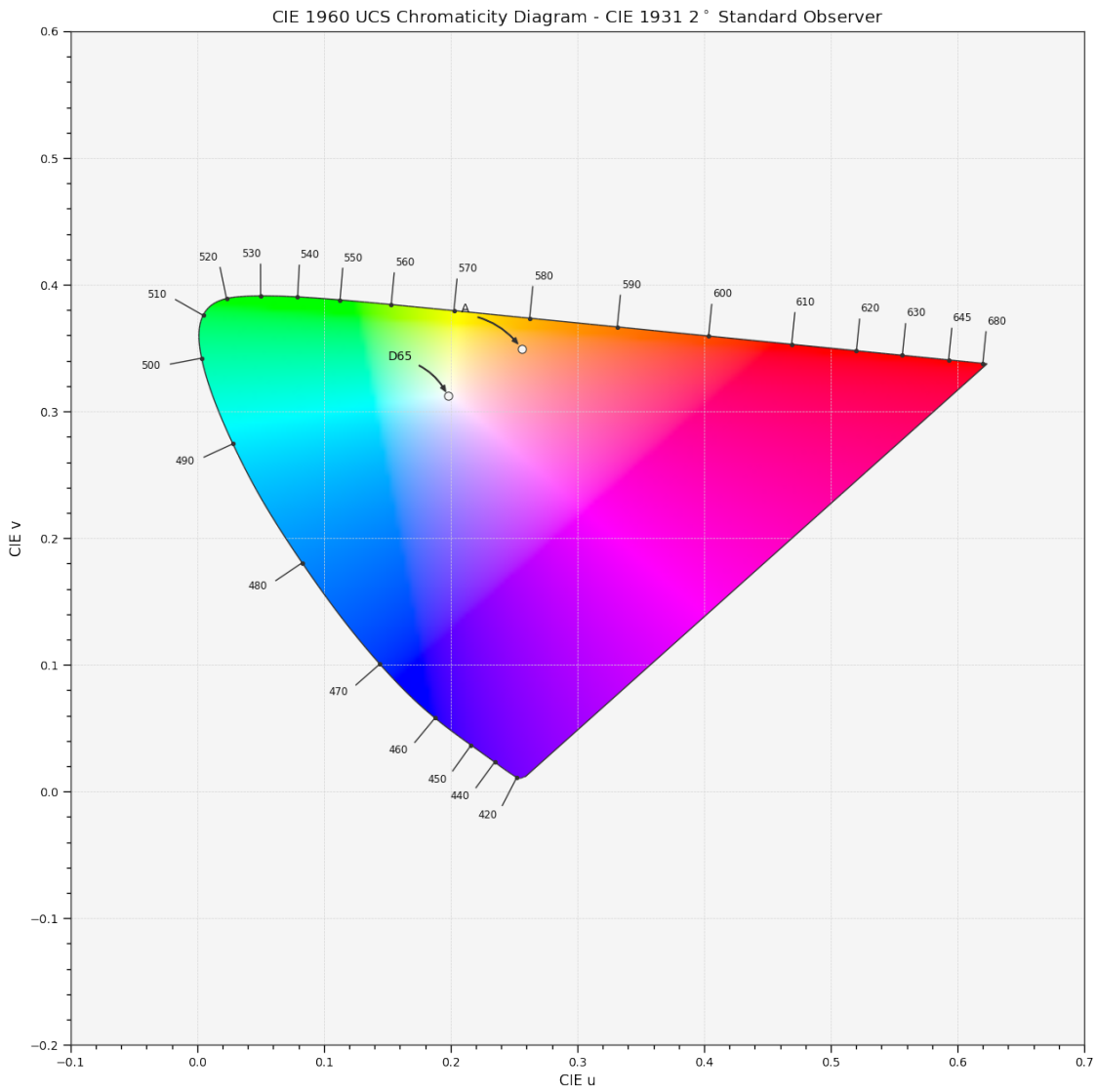
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1960UCS([A, D65])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_sds_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_sds_in_chromaticity_diagram_CIE1976UCS**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate_parameters=None*, *chromaticity_diagram_callable_CIE1976UCS=<function plot_chromaticity_diagram_CIE1976UCS>*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.

- **annotate_parameters** (dict or array_like, optional) – Parameters for the `plt.annotate()` definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if `annotate` is set to *True*. `annotate_parameters` can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

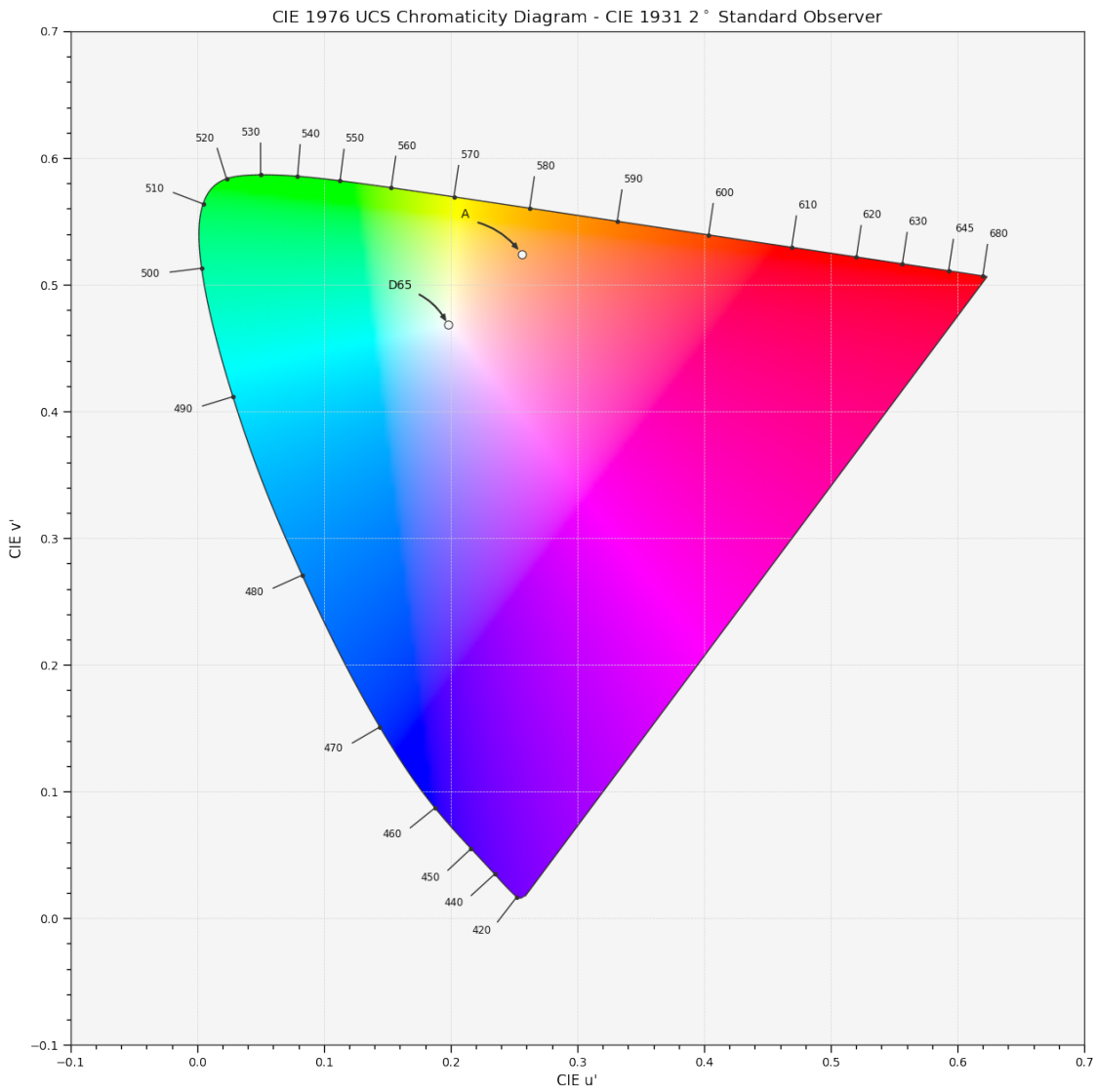
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.diagrams.plot_chromaticity_diagram()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram_CIE1976UCS([A, D65])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Ancillary Objects

`colour.plotting.diagrams`

plot_spectral_locus([cmfs, ...])	Plots the *Spectral Locus* according to given method.
plot_chromaticity_diagram_colours([samples, ...])	Plots the *Chromaticity Diagram* colours according to given method.
plot_chromaticity_diagram([cmfs, ...])	Plots the *Chromaticity Diagram* according to given method.
plot_sds_in_chromaticity_diagram(sds[, ...])	Plots given spectral distribution chromaticity coordinates into the *Chromaticity Diagram* using given method.

colour.plotting.diagrams.plot_spectral_locus

colour.plotting.diagrams.**plot_spectral_locus**(*cmfs='CIE 1931 2 Degree Standard Observer'*, *spectral_locus_colours=None*, *spectral_locus_labels=None*, *method='CIE 1931'*, ***kwargs*)

Plots the *Spectral Locus* according to given method.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions defining the *Spectral Locus*.
- **spectral_locus_colours** (array_like or unicode, optional) – *Spectral Locus* colours, if spectral_locus_colours is set to *RGB*, the colours will be computed according to the corresponding chromaticity coordinates.
- **spectral_locus_labels** (array_like, optional) – Array of wavelength labels used to customise which labels will be drawn around the spectral locus. Passing an empty array will result in no wavelength labels being drawn.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.

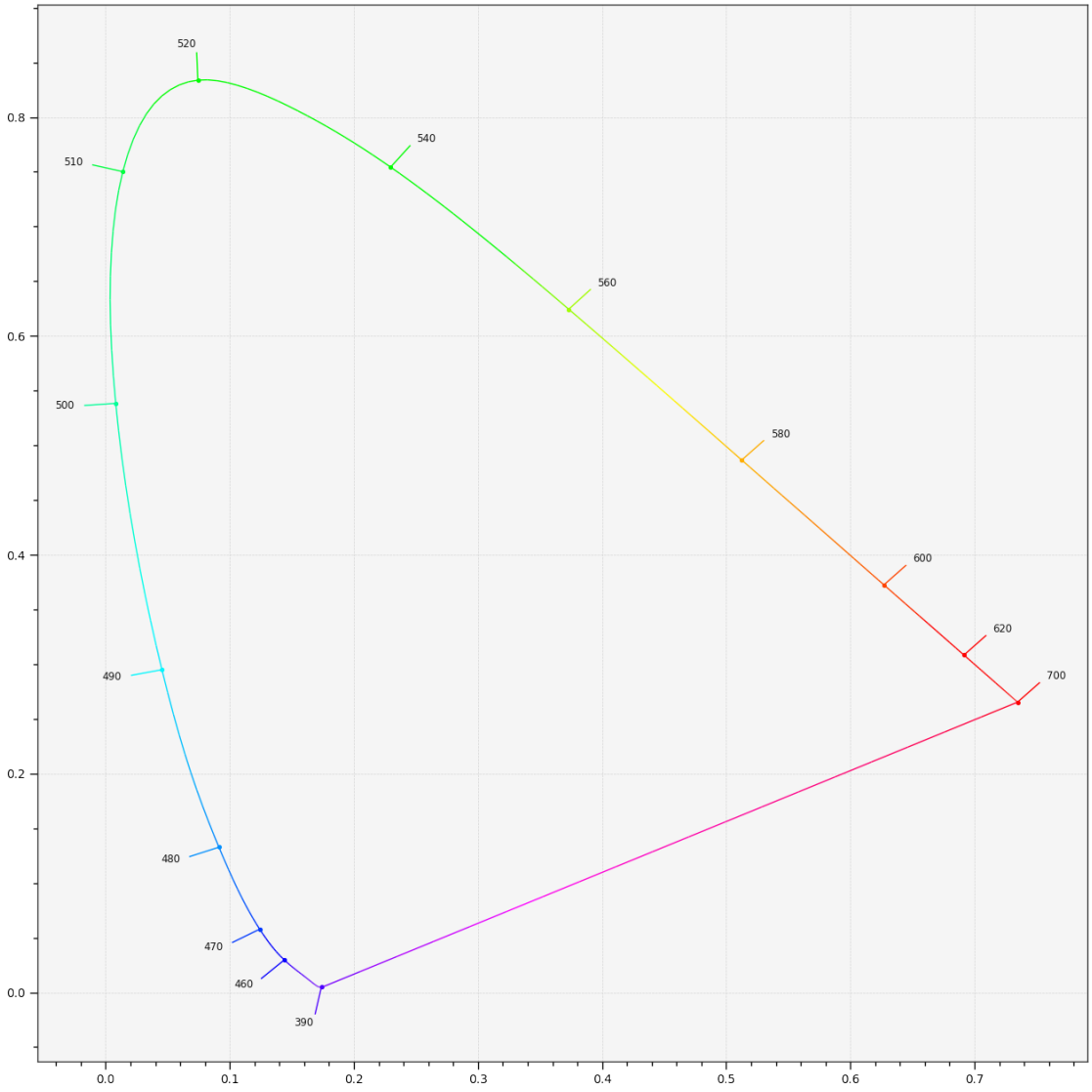
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_spectral_locus(spectral_locus_colours='RGB')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.diagrams.plot_chromaticity_diagram_colours

colour.plotting.diagrams.**plot_chromaticity_diagram_colours**(*samples=256*, *diagram_opacity=1.0*, *diagram_clipping_path=None*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *method='CIE 1931'*, ***kwargs*)

Plots the *Chromaticity Diagram* colours according to given method.

Parameters

- **samples** (numeric, optional) – Samples count on one axis.
- **diagram_opacity** (numeric, optional) – Opacity of the *Chromaticity Diagram* colours.
- **diagram_clipping_path** (array_like, optional) – Path of points used to clip the *Chromaticity Diagram* colours.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.

- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram_colours()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

colour.plotting.diagrams.plot_chromaticity_diagram

colour.plotting.diagrams.**plot_chromaticity_diagram**(*cmfs='CIE 1931 2 Degree Standard Observer'*, *show_diagram_colours=True*, *show_spectral_locus=True*, *method='CIE 1931'*, ***kwargs*)

Plots the *Chromaticity Diagram* according to given method.

Parameters

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **show_diagram_colours** (bool, optional) – Whether to display the *Chromaticity Diagram* background colours.
- **show_spectral_locus** (bool, optional) – Whether to display the *Spectral Locus*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.

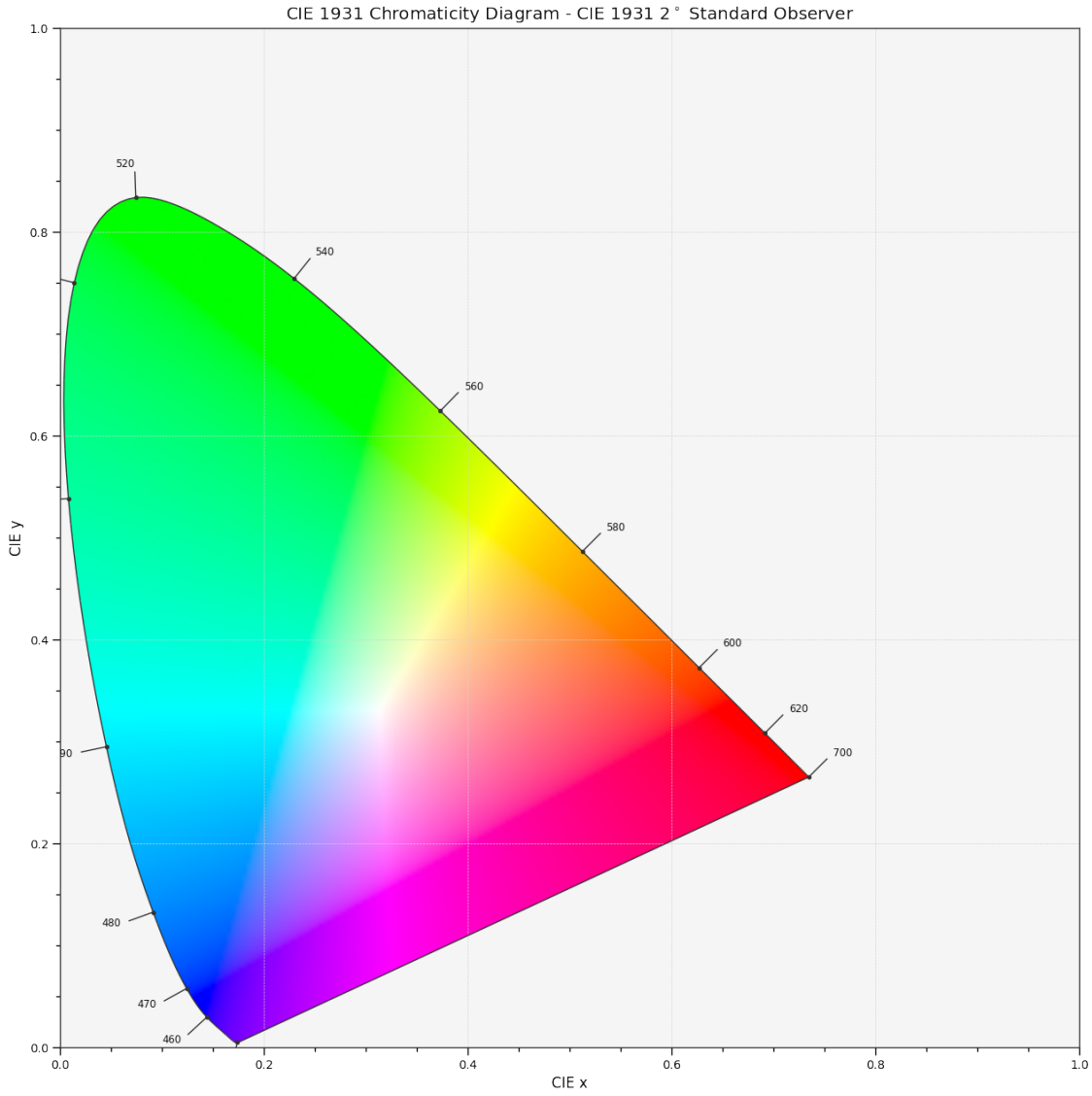
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_spectral_locus(), colour.plotting.diagrams.plot_chromaticity_diagram_colours(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_chromaticity_diagram()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.diagrams.plot_sds_in_chromaticity_diagram

colour.plotting.diagrams.**plot_sds_in_chromaticity_diagram**(*sds*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *annotate_parameters=None*, *chromaticity_diagram_callable=<function plot_chromaticity_diagram>*, *method='CIE 1931'*, ***kwargs*)

Plots given spectral distribution chromaticity coordinates into the *Chromaticity Diagram* using given method.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.

- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective spectral distribution names if annotate is set to *True*. annotate_parameters can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each spectral distribution.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.

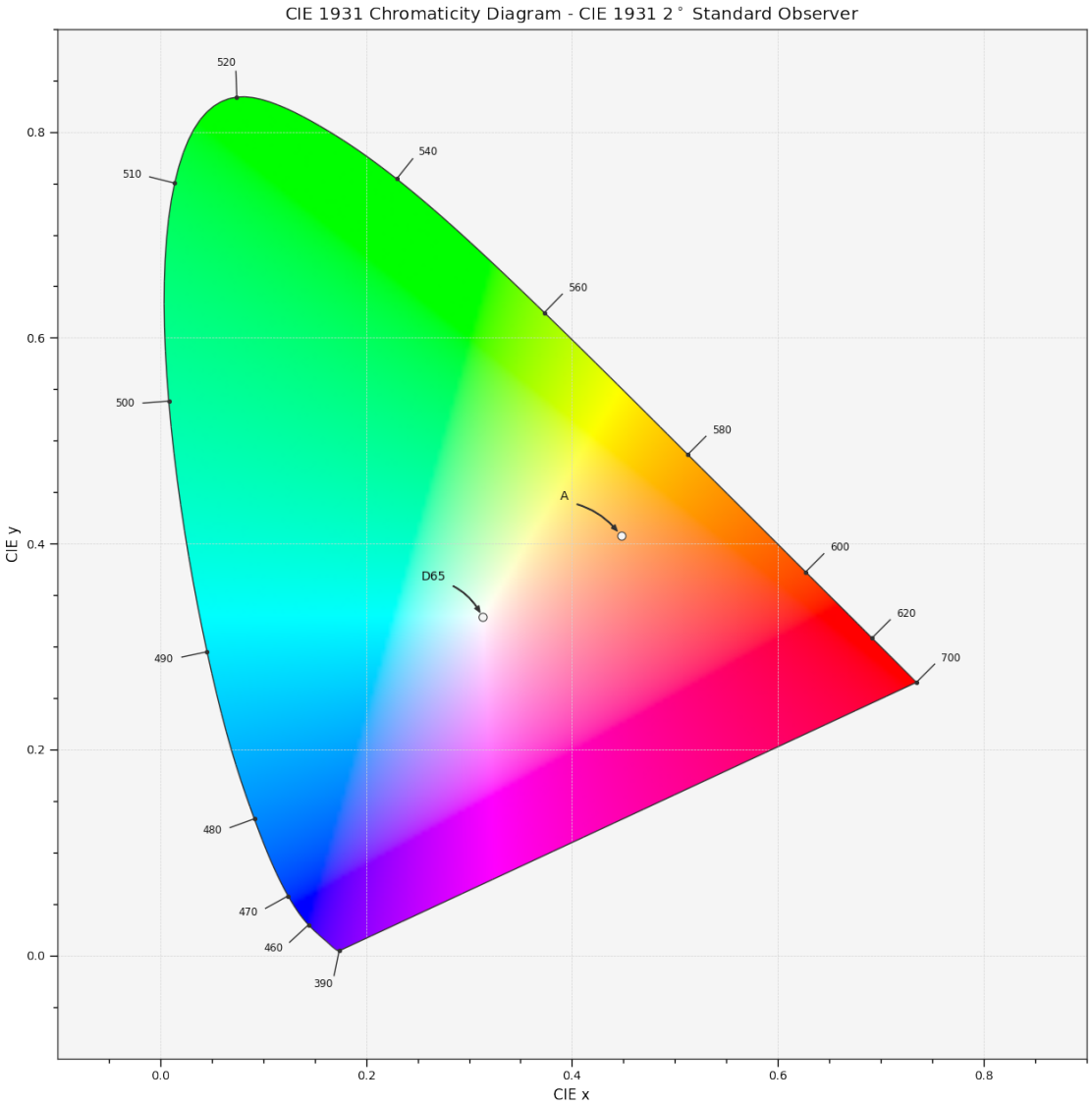
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> A = ILLUMINANTS_SDS['A']
>>> D65 = ILLUMINANTS_SDS['D65']
>>> plot_sds_in_chromaticity_diagram([A, D65])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Models

`colour.plotting`

plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931([...])	Plots given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS([...])	Plots given *RGB* colourspaces in the *CIE 1960 UCS Chromaticity Diagram*.
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS([...])	Plots given *RGB* colourspaces in the *CIE 1976 UCS Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(RGB)	Plots given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(RGB)	Plots given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(RGB)	Plots given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram([...])	Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1931 Chromaticity Diagram*.

Continued on next page

Table 212 – continued from previous page

plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS([...])	Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1960 UCS Chromaticity Diagram*.
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS([...])	Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1976 UCS Chromaticity Diagram*.
plot_single_cctf([cctf, cctf_decoding])	Plots given colourspace colour component transfer function.
plot_multi_cctfs([cctfs, cctf_decoding])	Plots given colour component transfer functions.
plot_constant_hue_loci(data, model[, ...])	Plots given constant hue loci colour matches data such as that from [HB95] or [EF98] that are easily loaded with Colour - Datasets.

colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931**(*colourspaces=None*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1931=<functio plot_chromaticity_diagram_CIE1931>*, ***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS**(*colourspaces=None*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1960UCS=<plot_chromaticity_diagram_CIE1960U*, ***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.

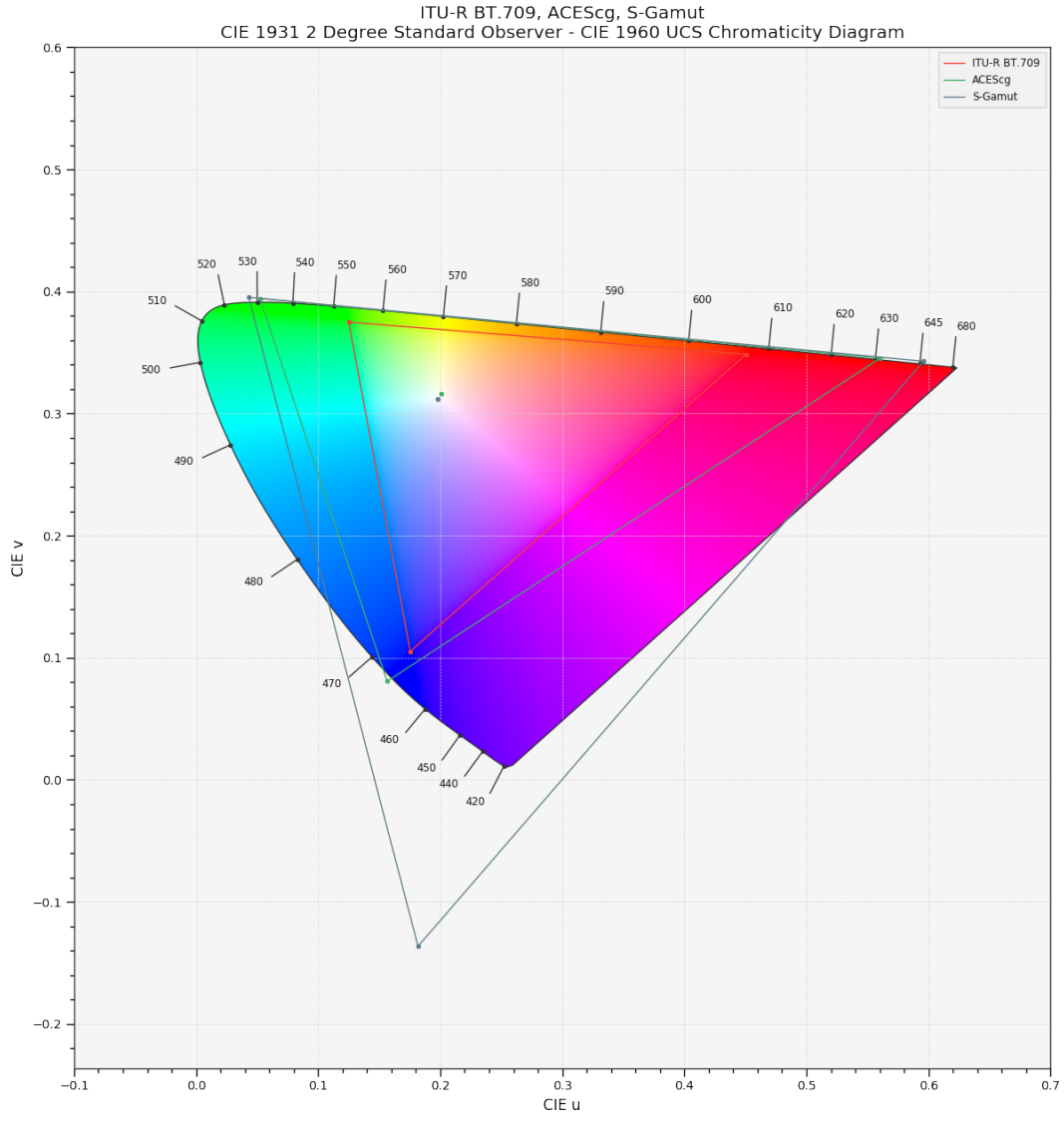
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS**(*colourspaces=None*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable_CIE1976UCS=<*
plot_chromaticity_diagram_CIE1976U
***kwargs*)

Plots given *RGB* colourspaces in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.

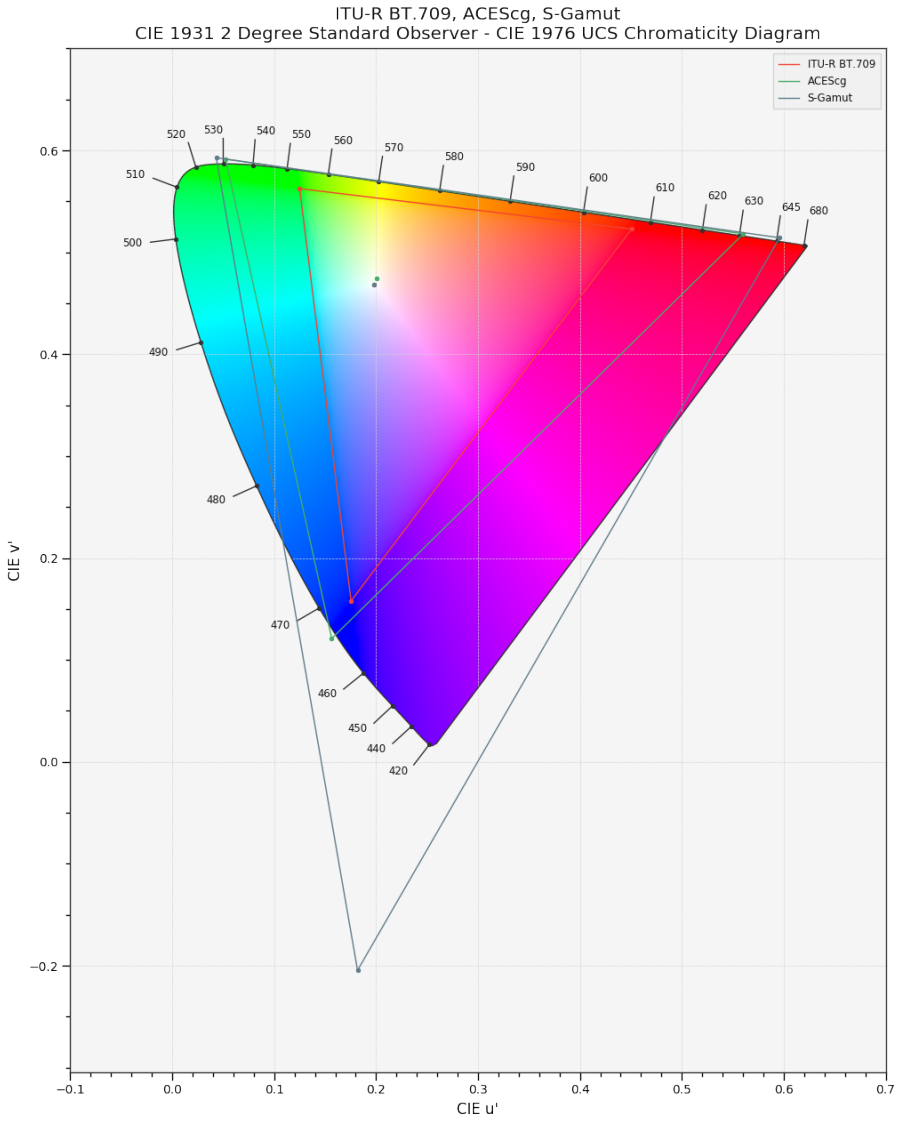
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931**(*RGB*, *colourspace='sRGB'*, *chromaticity_diagram_callable_CIE1931=<func*
plot_RGB_colourspaces_in_chromaticity
scatter_parameters=None, ***kwargs*)

Plots given *RGB* colourspace array in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **scatter_parameters** (dict, optional) – Parameters for the plt.scatter() definition, if c is set to *RGB*, the scatter will use given RGB colours.

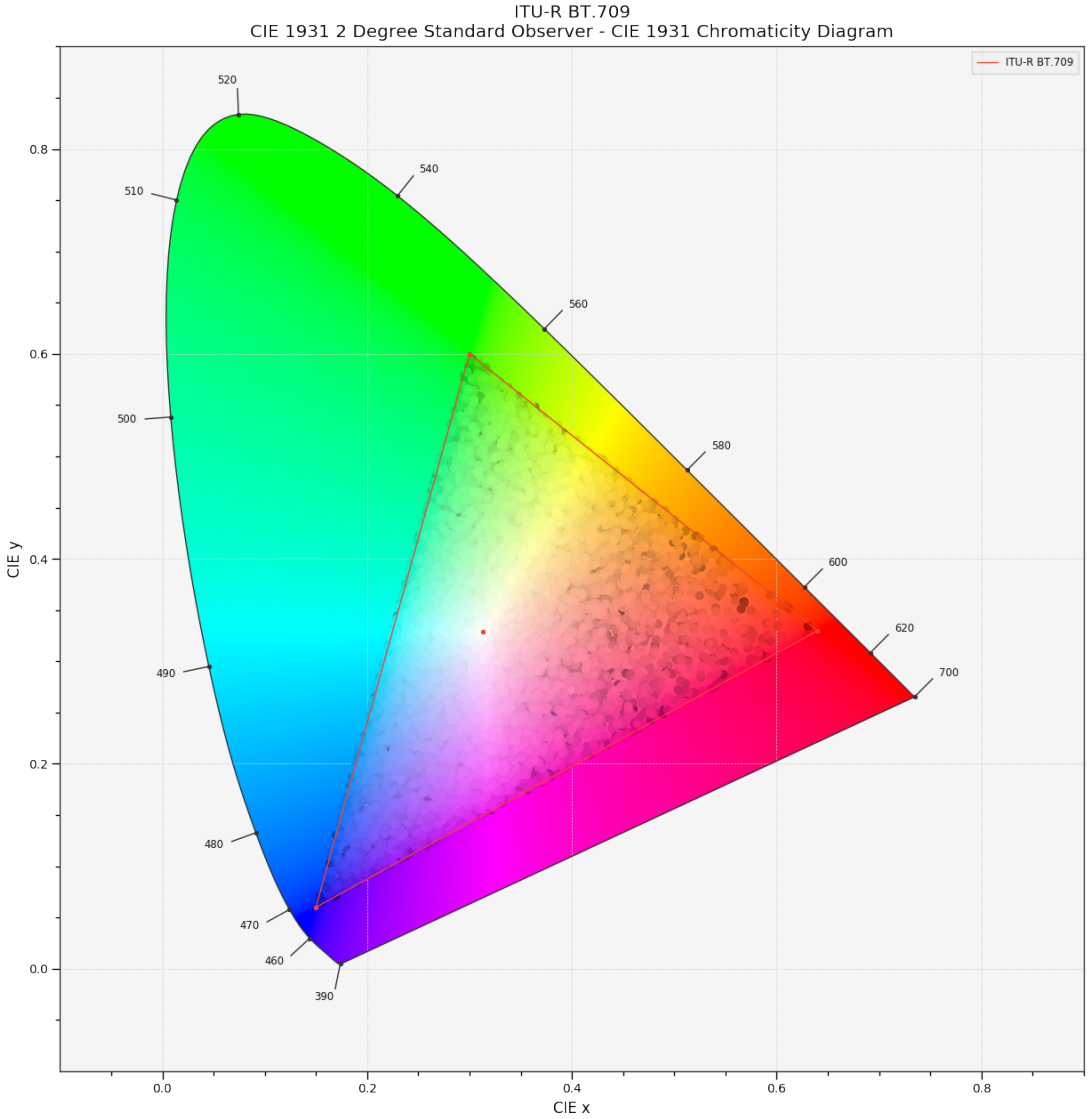
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.diagrams.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
...     RGB, 'ITU-R BT.709')
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS**(*RGB*, *colourspace='sRGB'*, *chromaticity_diagram_callable_CIE1960UCS=plot_RGB_colourspaces_in_chromat*, *scatter_parameters=None*, ***kwargs*)

Plots given *RGB* colourspace array in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **scatter_parameters** (dict, optional) – Parameters for the plt.scatter() definition, if c is set to *RGB*, the scatter will use given RGB colours.

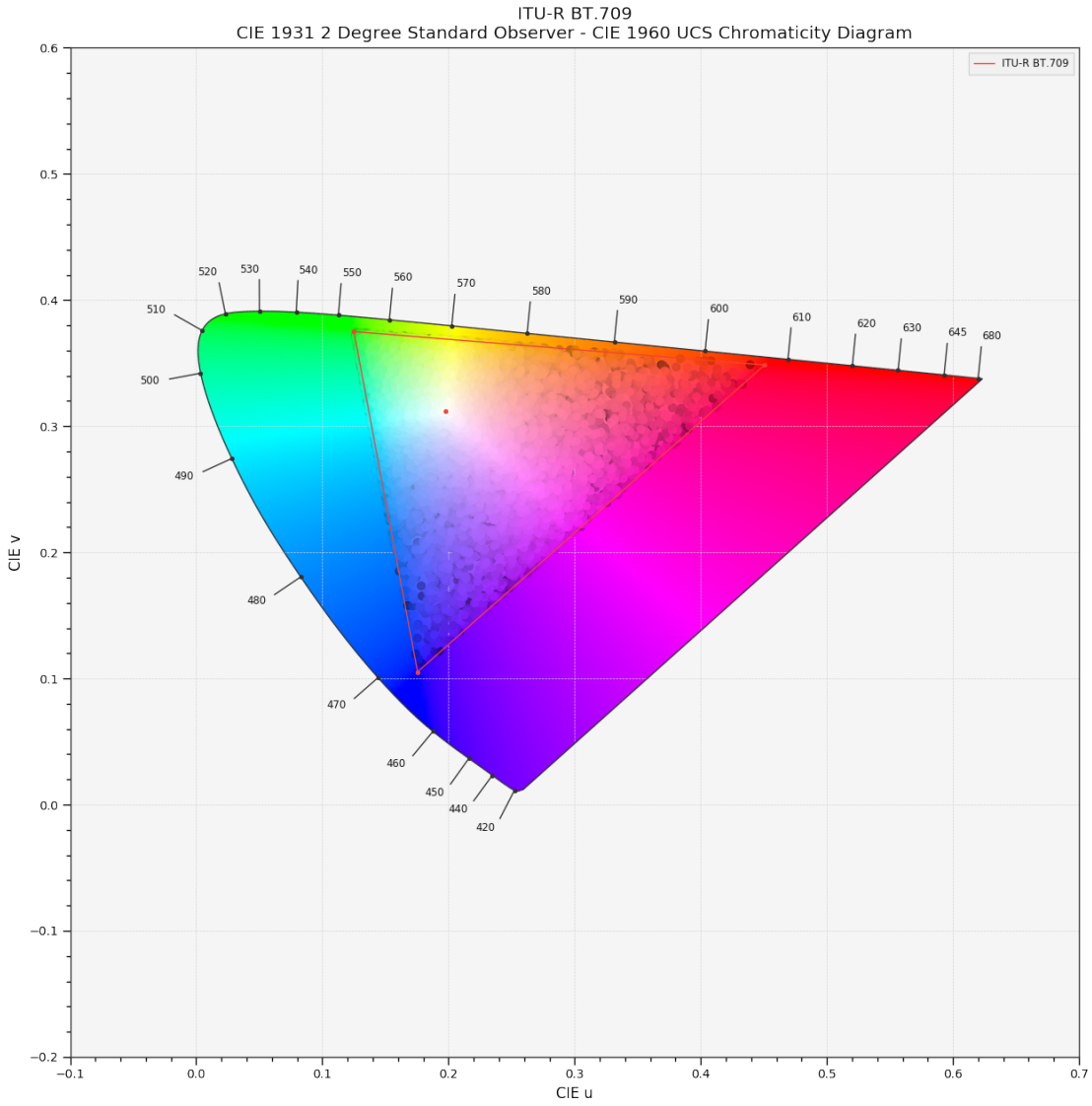
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.diagrams.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS(
...     RGB, 'ITU-R BT.709')
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS**(*RGB*, *colourspace='sRGB'*, *chromaticity_diagram_callable_CIE1976UCS=plot_RGB_colourspaces_in_chromat*, *scatter_parameters=None*, ***kwargs*)

Plots given *RGB* colourspace array in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **scatter_parameters** (dict, optional) – Parameters for the plt.scatter() definition, if c is set to *RGB*, the scatter will use given RGB colours.

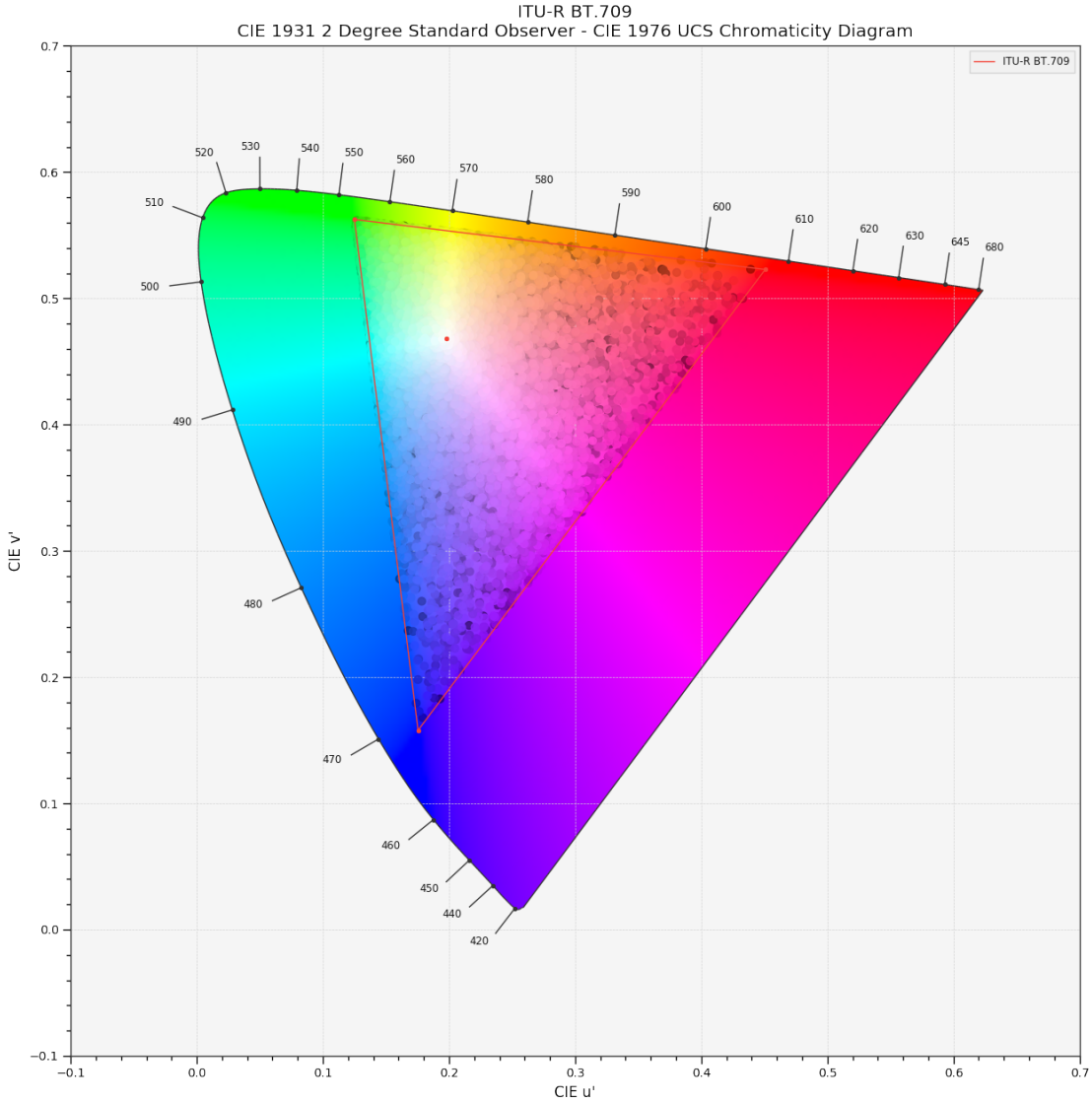
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.diagrams.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS(
...     RGB, 'ITU-R BT.709')
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931**(*chromaticity_diagram_callable_CIE1931=plot_chromaticity_diagram_CIE1931*, *chromaticity_diagram_clipping=False*, *ellipse_parameters=None*, ***kwargs*)

Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1931 Chromaticity Diagram*.

Parameters

- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool, optional,) – Whether to clip the *CIE 1931 Chromaticity Diagram* colours with the ellipses.
- **ellipse_parameters** (dict or array_like, optional) – Parameters for the Ellipse class, ellipse_parameters can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.

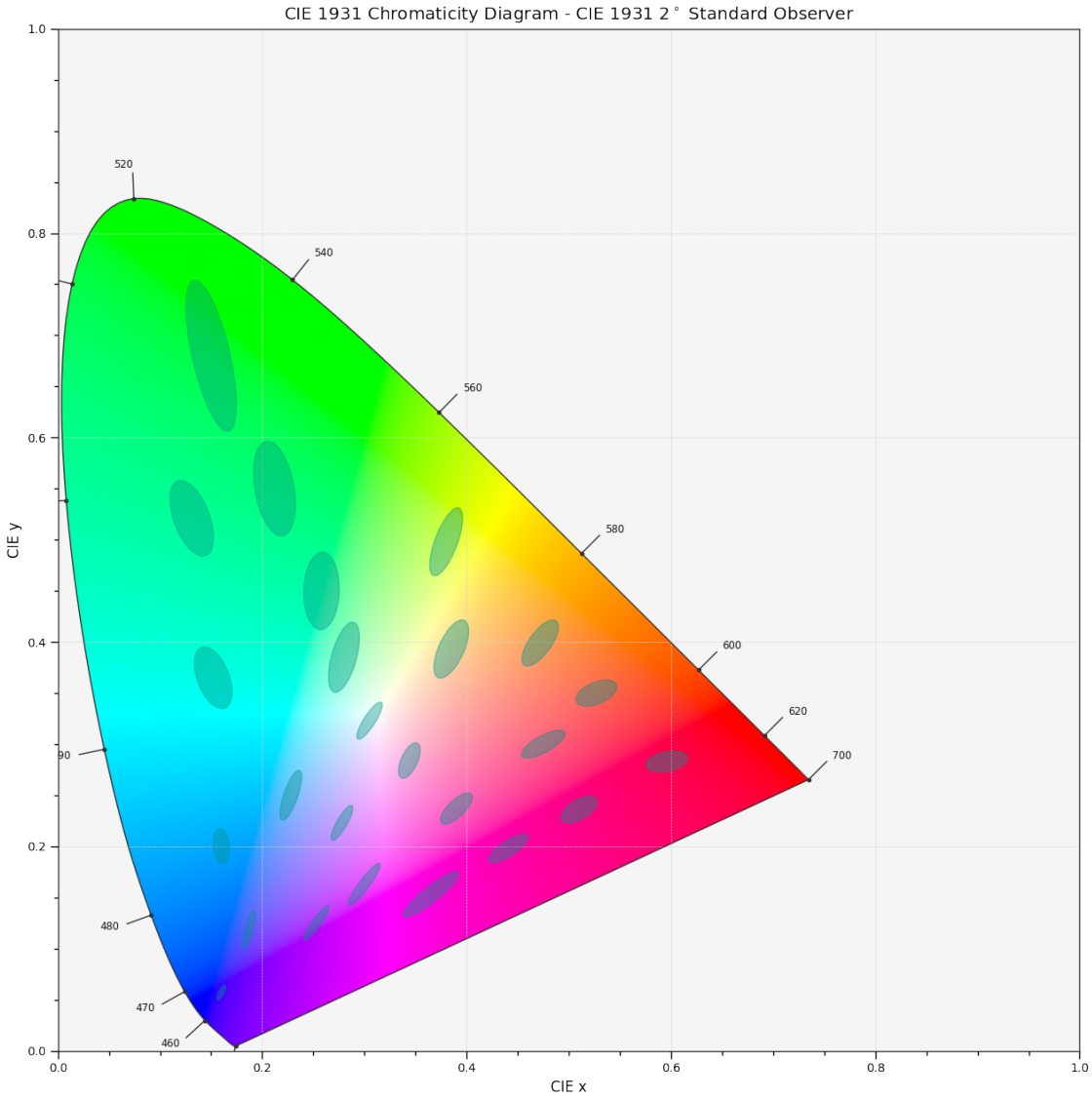
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931()
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS**(*chromaticity_diagram_callable_C*
plot_chromaticity_diagram_CIE1
chro-
matic-
ity_diagram_clipping=False,
el-
lipse_parameters=None,
***kwargs*)

Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool, optional,) – Whether to clip the *CIE 1960 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_parameters** (dict or array_like, optional) – Parameters for the Ellipse class, ellipse_parameters can be either a single dictionary applied to

all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.

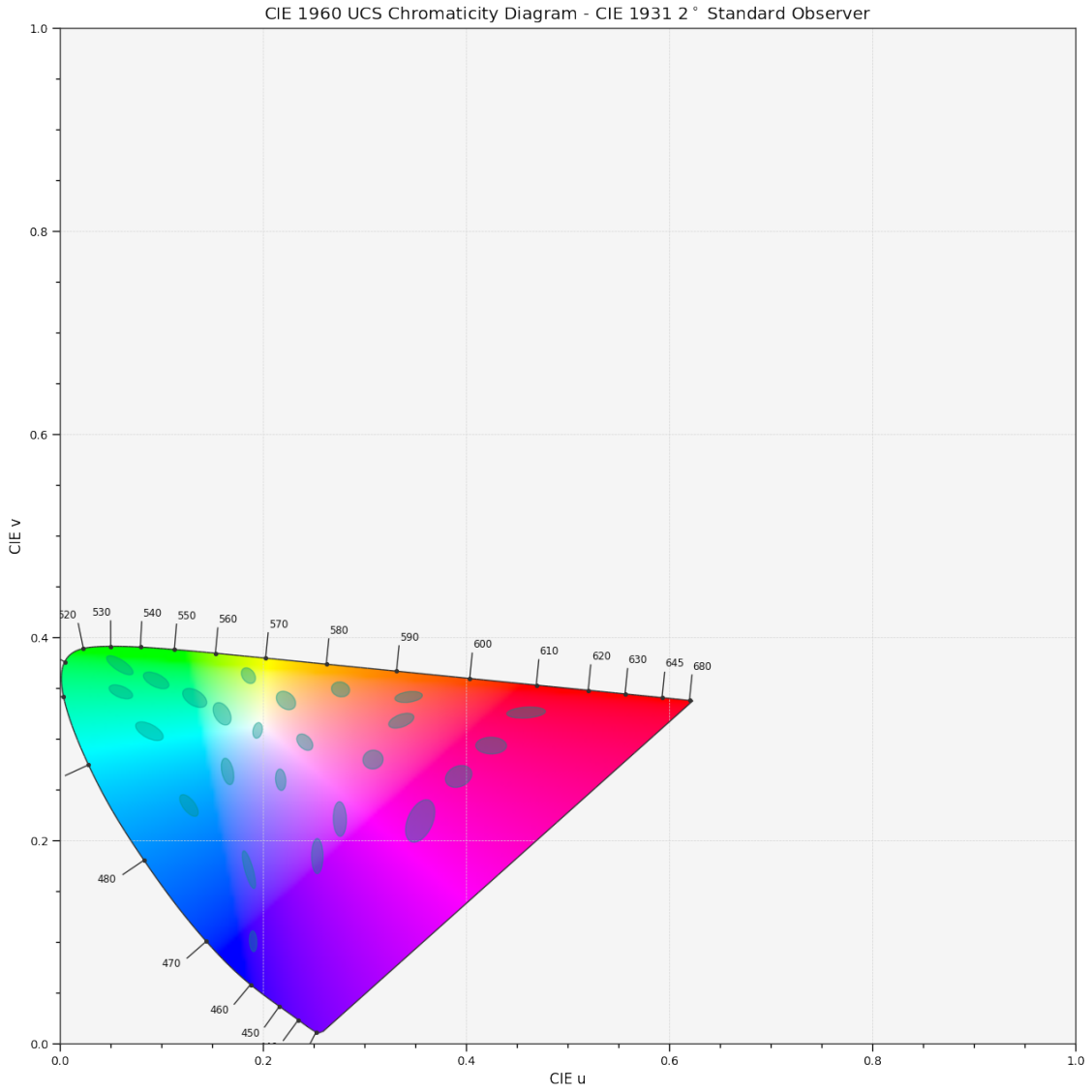
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS()
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS

colour.plotting.**plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS**(*chromaticity_diagram_callable_C*
plot_chromaticity_diagram_CIE1
chro-
matic-
ity_diagram_clipping=False,
el-
lipse_parameters=None,
***kwargs*)

Plots *MacAdam (1942) Ellipses (Observer PGN)* in the *CIE 1976 UCS Chromaticity Diagram*.

Parameters

- **chromaticity_diagram_callable_CIE1976UCS** (callable, optional) – Callable responsible for drawing the *CIE 1976 UCS Chromaticity Diagram*.
- **chromaticity_diagram_clipping** (bool, optional,) – Whether to clip the *CIE 1976 UCS Chromaticity Diagram* colours with the ellipses.
- **ellipse_parameters** (dict or array_like, optional) – Parameters for the Ellipse class, ellipse_parameters can be either a single dictionary applied to all the ellipses with same settings or a sequence of dictionaries with different settings for each ellipse.

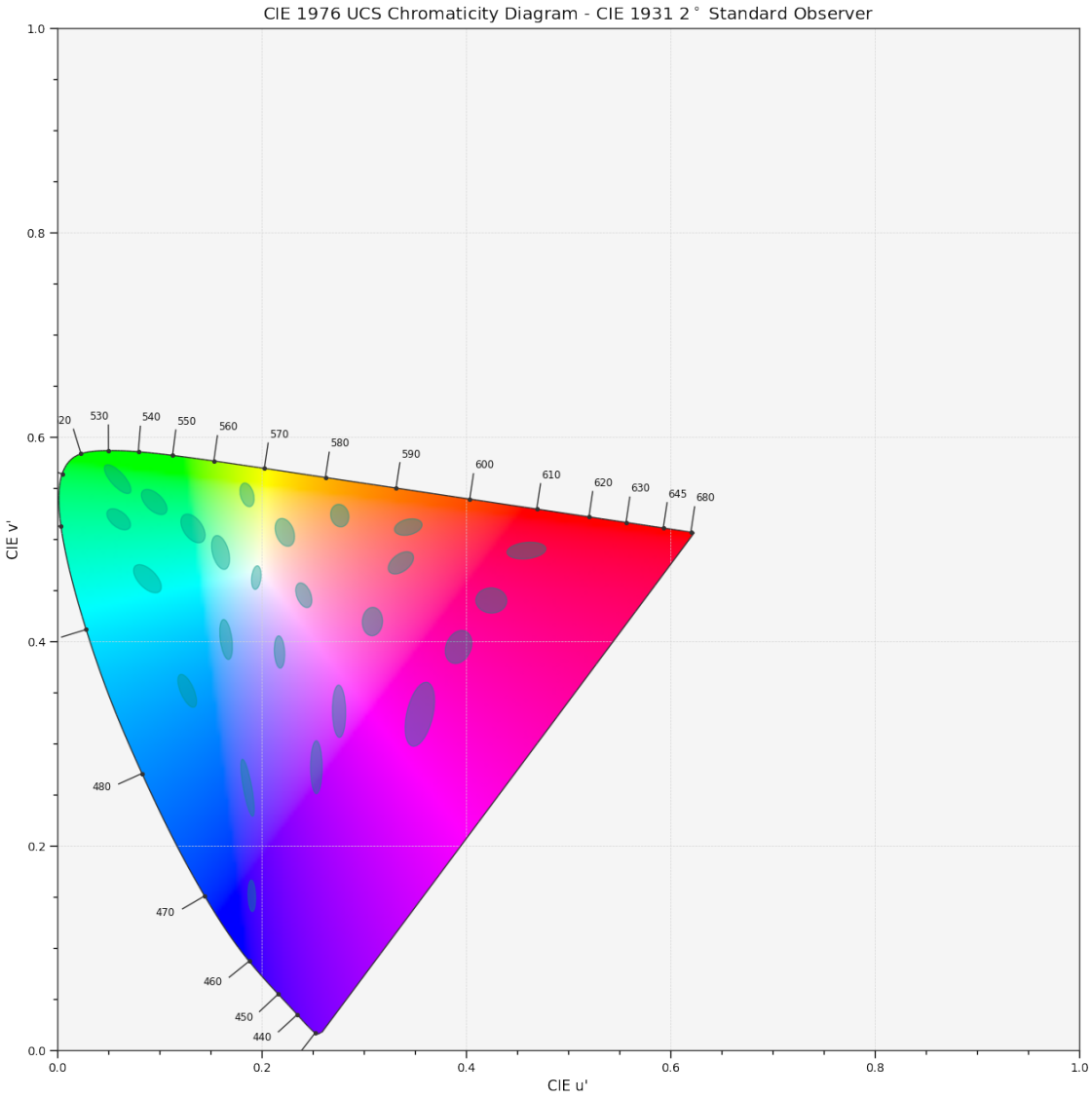
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.models.plot_ellipses_MacAdam1942_in_chromaticity_diagram()}, colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS()
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_cctf

colour.plotting.**plot_single_cctf**(*cctf='ITU-R BT.709'*, *cctf_decoding=False*, ***kwargs*)

Plots given colourspace colour component transfer function.

Parameters

- **cctf** (unicode, optional) – Colour component transfer function to plot.
- **cctf_decoding** (bool) – Plot the decoding colour component transfer function instead.

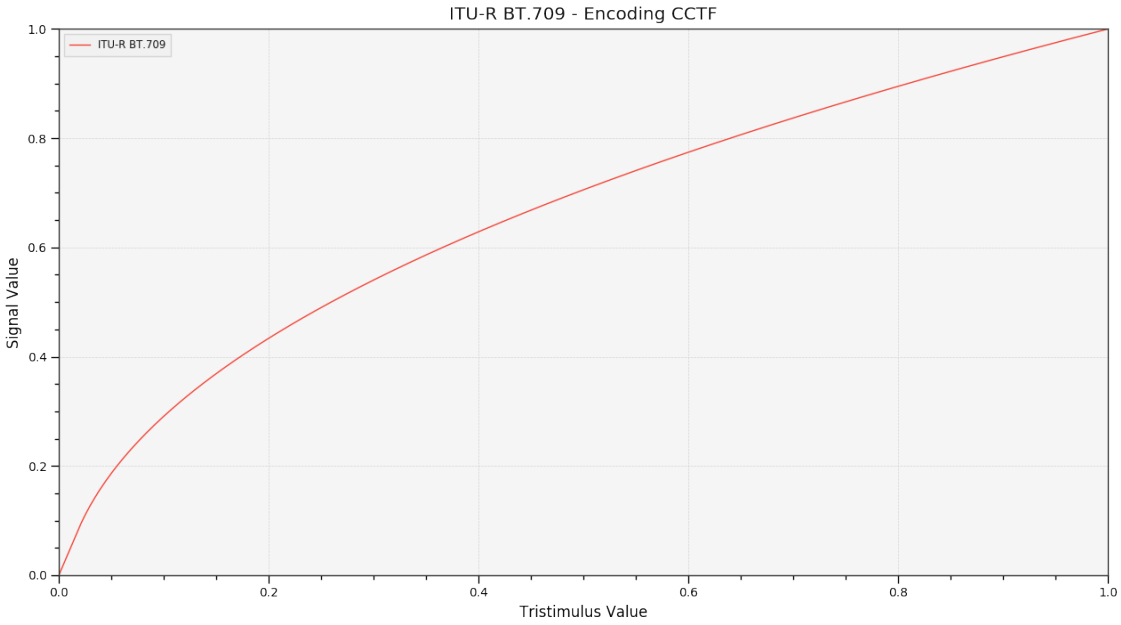
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_cctf('ITU-R BT.709')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_cctfs

colour.plotting.**plot_multi_cctfs**(*cctfs=None*, *cctf_decoding=False*, ***kwargs*)

Plots given colour component transfer functions.

Parameters

- **cctfs** (array_like, optional) – Colour component transfer function to plot.
- **cctf_decoding** (bool) – Plot the decoding colour component transfer function instead.

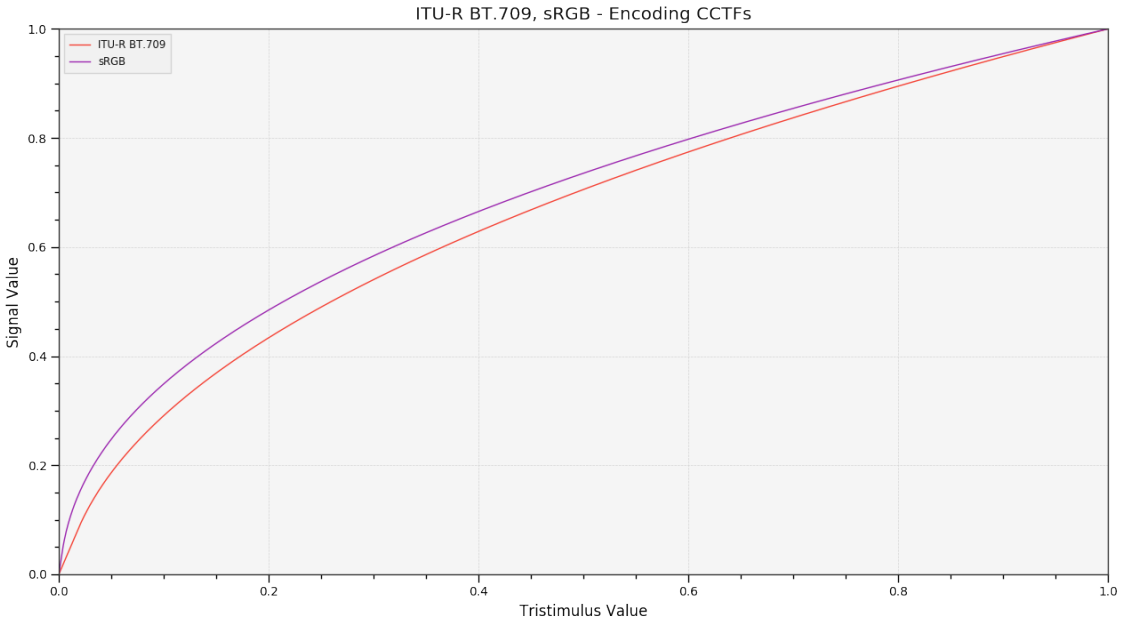
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_cctfs(['ITU-R BT.709', 'sRGB'])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_constant_hue_loci

colour.plotting.**plot_constant_hue_loci**(*data*, *model*, *scatter_parameters=None*, ***kwargs*)

Plots given constant hue loci colour matches data such as that from [HB95] or [EF98] that are easily loaded with Colour - Datasets.

Parameters

- **data** (array_like) – Constant hue loci colour matches data expected to be an *array_like* as follows:

```
[
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...), {metadata}),
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...), {metadata}),
    ('name', XYZ_r, XYZ_cr, (XYZ_ct, XYZ_ct, XYZ_ct, ...), {metadata}),
    ...
]
```

  where name is the hue angle or name, XYZ_r the *CIE XYZ* tristimulus values of the reference illuminant, XYZ_cr the *CIE XYZ* tristimulus values of the reference colour under the reference illuminant, XYZ_ct the *CIE XYZ* tristimulus values of the colour matches under the reference illuminant and metadata the dataset metadata.

- **model** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE xy', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE Luv uv', 'CIE LCHuv', 'CIE UCS', 'CIE UCS uv', 'CIE UVW', 'DIN 99', 'Hunter Lab', 'Hunter Rdab', 'IPT', 'JzAzBz', 'OSA UCS', 'hdr-CIELAB', 'hdr-IPT'}**, Colourspace model.
- **scatter_parameters** (dict, optional) – Parameters for the plt.scatter() definition, if c is set to *RGB*, the scatter will use given RGB colours.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

References

[EF98], [HB95], [Man19]

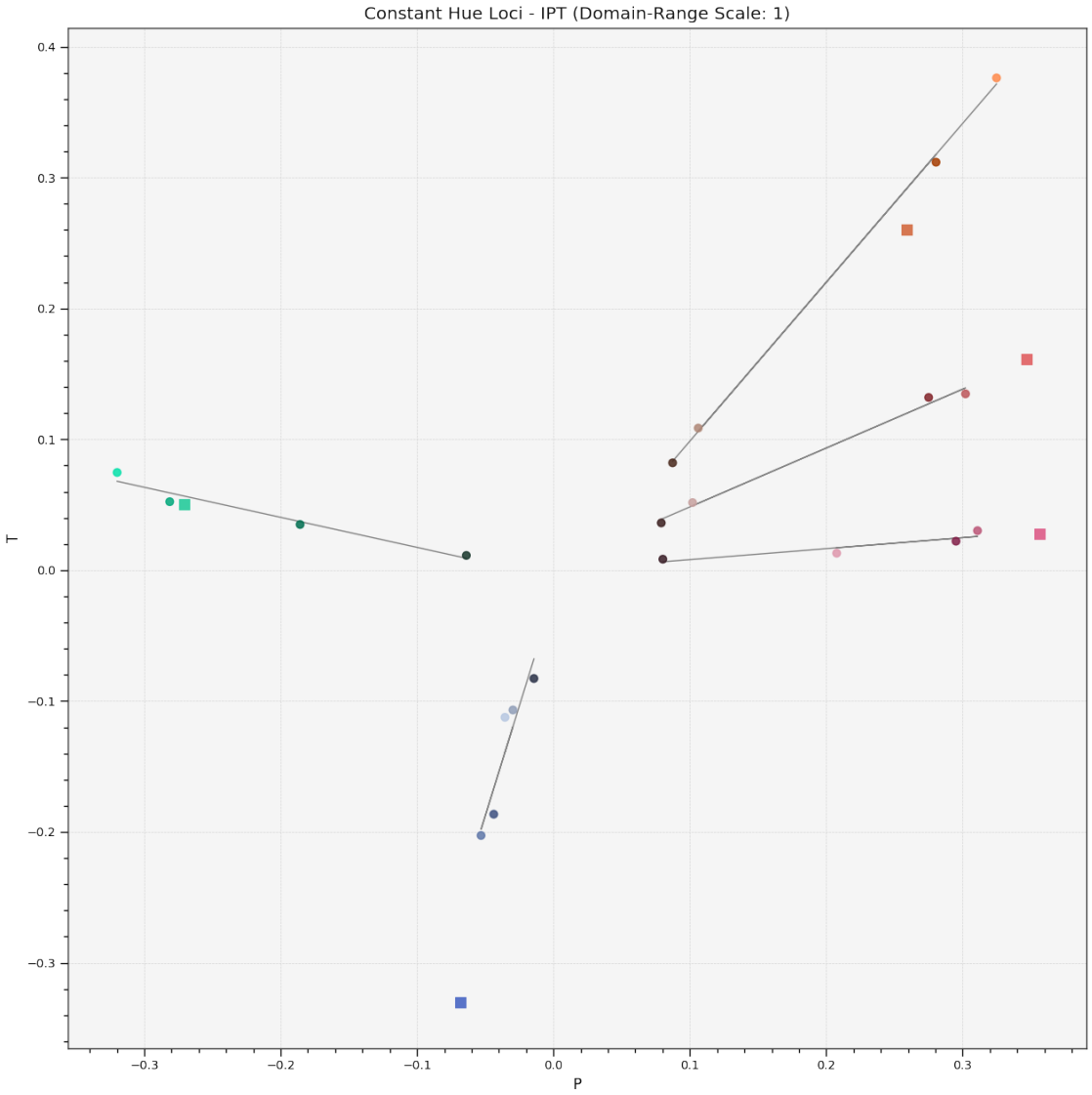
Examples

```
>>> data = np.array([
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.40920000, 0.28120000, 0.30600000]),
...         np.array([
...             [0.02495100, 0.01908600, 0.02032900],
...             [0.10944300, 0.06235900, 0.06788100],
...             [0.27186500, 0.18418700, 0.19565300],
...             [0.48898900, 0.40749400, 0.44854600],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.30760000, 0.48280000, 0.42770000]),
...         np.array([
...             [0.02108000, 0.02989100, 0.02790400],
...             [0.06194700, 0.11251000, 0.09334400],
...             [0.15255800, 0.28123300, 0.23234900],
...             [0.34157700, 0.56681300, 0.47035300],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.39530000, 0.28120000, 0.18450000]),
...         np.array([
...             [0.02436400, 0.01908600, 0.01468800],
...             [0.10331200, 0.06235900, 0.02854600],
...             [0.26311900, 0.18418700, 0.12109700],
...             [0.43158700, 0.40749400, 0.39008600],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.20510000, 0.18420000, 0.57130000]),
...         np.array([
...             [0.03039800, 0.02989100, 0.06123300],
...             [0.08870000, 0.08498400, 0.21843500],
...             [0.18405800, 0.18418700, 0.40111400],
...             [0.32550100, 0.34047200, 0.50296900],
...             [0.53826100, 0.56681300, 0.80010400],
...         ]),
...         None,
...     ],
...     [
...         None,
...         np.array([0.95010000, 1.00000000, 1.08810000]),
...         np.array([0.35770000, 0.28120000, 0.11250000]),
...         np.array([
```

(continues on next page)

(continued from previous page)

```
...             [0.03678100, 0.02989100, 0.01481100],
...             [0.17127700, 0.11251000, 0.01229900],
...             [0.30080900, 0.28123300, 0.21229800],
...             [0.52976000, 0.40749400, 0.11720000],
...         ]),
...         None,
...     ],
... ])
>>> plot_constant_hue_loci(data, 'IPT')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Ancillary Objects

`colour.plotting.models`

`common_colourspace_model_axis_reorder`(a[, model])	Reorder the axes of given colourspace model a array according to the most common volume plotting axes order.
`plot_pointer_gamut`([method])	Plots *Pointer's Gamut* according to given method.

Continued on next page

Table 213 – continued from previous page

plot_RGB_colourspaces_in_chromaticity_diagram[...]	Plots given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.
plot_RGB_chromaticities_in_chromaticity_diagram[...]	Plots given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.

colour.plotting.models.common_colourspace_model_axis_reorder

colour.plotting.models.**common_colourspace_model_axis_reorder**(*a*, *model=None*)

Reorder the axes of given colourspace model a array according to the most common volume plotting axes order.

Parameters

- **a** (array_like) – Colourspace model a array.
- **model** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE xy', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE Luv uv', 'CIE LCHuv', 'CIE UCS', 'CIE UCS uv', 'CIE UVW', 'DIN 99', 'Hunter Lab', 'Hunter Rdab', 'IPT', 'JzAzBz', 'OSA UCS', 'hdr-CIELAB', 'hdr-IPT'}**, Colourspace model.

Returns Reordered colourspace model a array.

Return type ndarray

Examples

```
>>> a = np.array([0, 1, 2])
>>> common_colourspace_model_axis_reorder(a)
array([0, 1, 2])
>>> common_colourspace_model_axis_reorder(a, 'CIE Lab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'CIE LCHab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'CIE Luv')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'CIE LCHab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'DIN 99')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'Hunter Lab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'Hunter Rdab')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'IPT')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'JzAzBz')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'OSA UCS')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'hdr-CIELAB')
array([ 1.,  2.,  0.])
>>> common_colourspace_model_axis_reorder(a, 'hdr-IPT')
array([ 1.,  2.,  0.])
```

colour.plotting.models.plot_pointer_gamut

colour.plotting.models.**plot_pointer_gamut**(*method='CIE 1931'*, ***kwargs*)

Plots *Pointer's Gamut* according to given method.

Parameters **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, Plotting method.

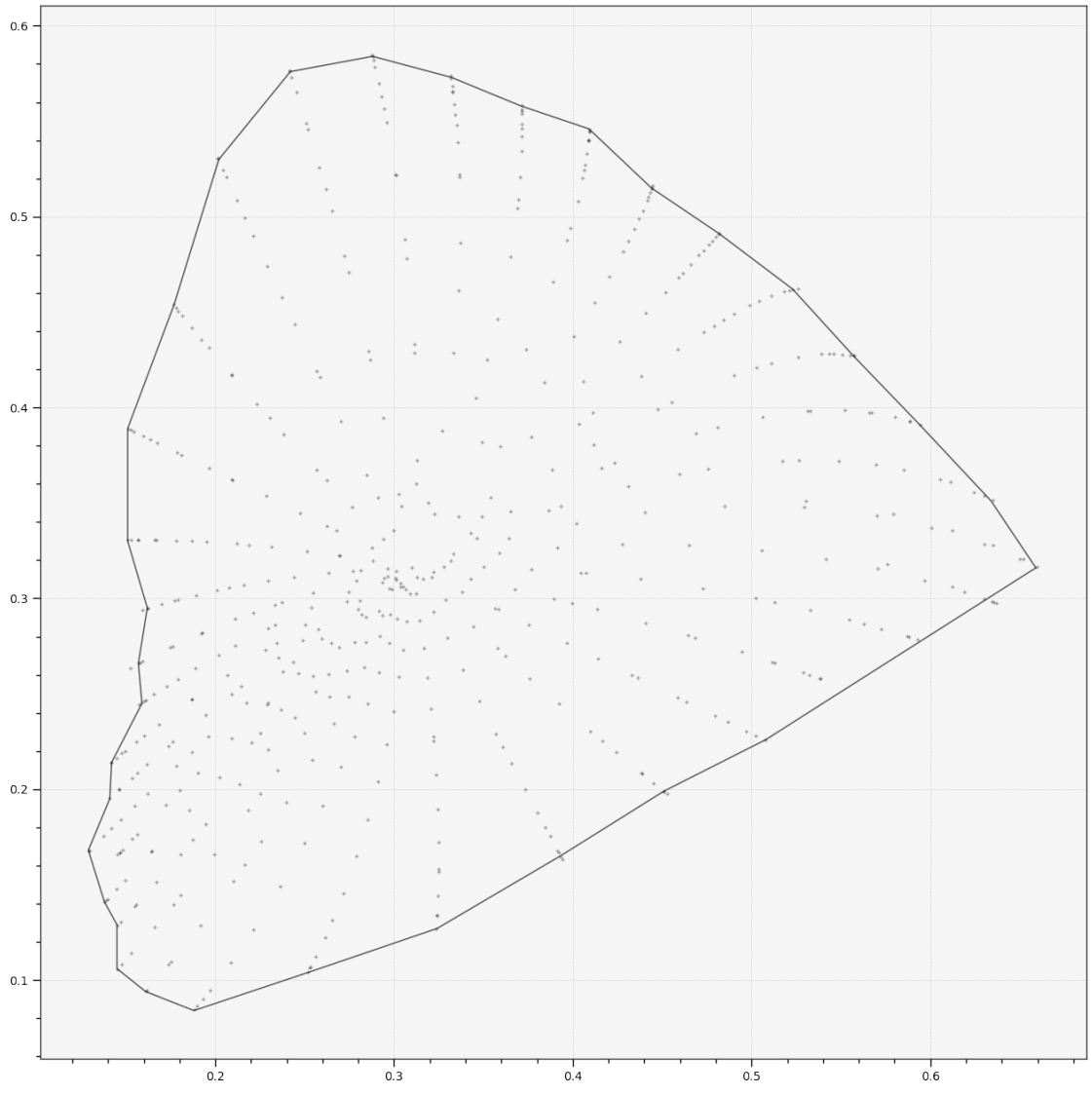
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_pointer_gamut()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.models.plot_RGB_colourspaces_in_chromaticity_diagram

colour.plotting.models.**plot_RGB_colourspaces_in_chromaticity_diagram**(*colourspaces=None*, *cmfs='CIE 1931 2 Degree Standard Observer'*, *chromaticity_diagram_callable=<function plot_chromaticity_diagram>*, *method='CIE 1931'*, *show_whitepoints=True*, *show_pointer_gamut=False*, ***kwargs*)

Plots given *RGB* colourspaces in the *Chromaticity Diagram* according to given method.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for *Chromaticity Diagram* bounds.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.
- **show_whitepoints** (bool, optional) – Whether to display the *RGB* colourspaces whitepoints.
- **show_pointer_gamut** (bool, optional) – Whether to display the *Pointer's Gamut*.

Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.plot_pointer_gamut(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_in_chromaticity_diagram(
...     ['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```

colour.plotting.models.plot_RGB_chromaticities_in_chromaticity_diagram

colour.plotting.models.**plot_RGB_chromaticities_in_chromaticity_diagram**(*RGB*, *colourspace='sRGB'*, *chromaticity_diagram_callable=<function plot_RGB_colourspaces_in_chromaticity_*, *method='CIE 1931'*, *scatter_parameters=None*, ***kwargs*)

Plots given *RGB* colourspace array in the *Chromaticity Diagram* according to given method.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (optional, unicode) – *RGB* colourspace of the *RGB* array.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.
- **scatter_parameters** (dict, optional) – Parameters for the plt.scatter() definition, if c is set to *RGB*, the scatter will use given RGB colours.

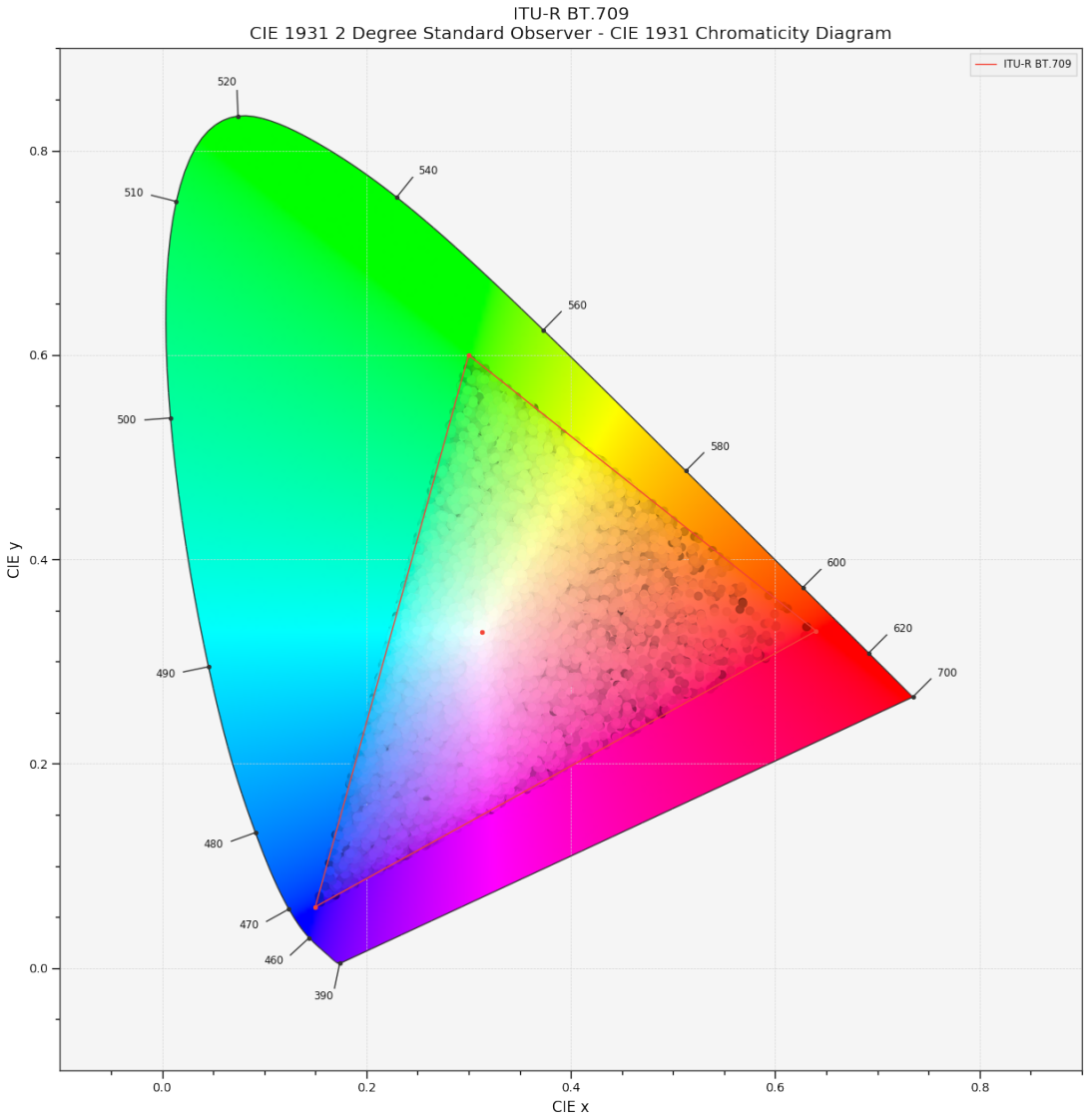
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.diagrams.plot_RGB_colourspaces_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram(
...     RGB, 'ITU-R BT.709')
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Notation Systems

`colour.plotting`

`plot_single_munsell_value_function`([function])	Plots given *Lightness* function.
`plot_multi_munsell_value_functions`([functions])	Plots given *Munsell* value functions.

colour.plotting.plot_single_munsell_value_function

`colour.plotting.`**`plot_single_munsell_value_function`**(*function='ASTM D1535'*, ***kwargs*)

Plots given *Lightness* function.

Parameters **`function`** (`unicode`, `optional`) – *Munsell* value function to plot.

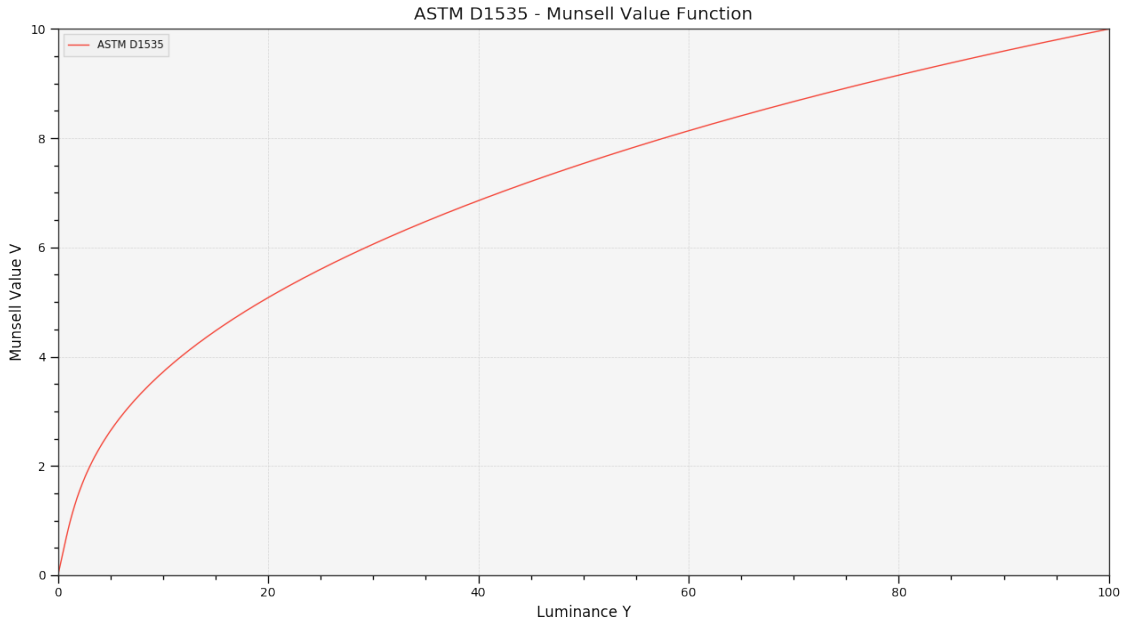
Other Parameters ****kwargs** (*dict, optional*) – {`colour.plotting.artist()`, `colour.plotting.plot_multi_functions()`, `colour.plotting.render()`}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_munsell_value_function('ASTM D1535')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_munsell_value_functions

colour.plotting.**plot_multi_munsell_value_functions**(*functions=None*, ***kwargs*)

Plots given *Munsell* value functions.

Parameters **functions** (array_like, optional) – *Munsell* value functions to plot.

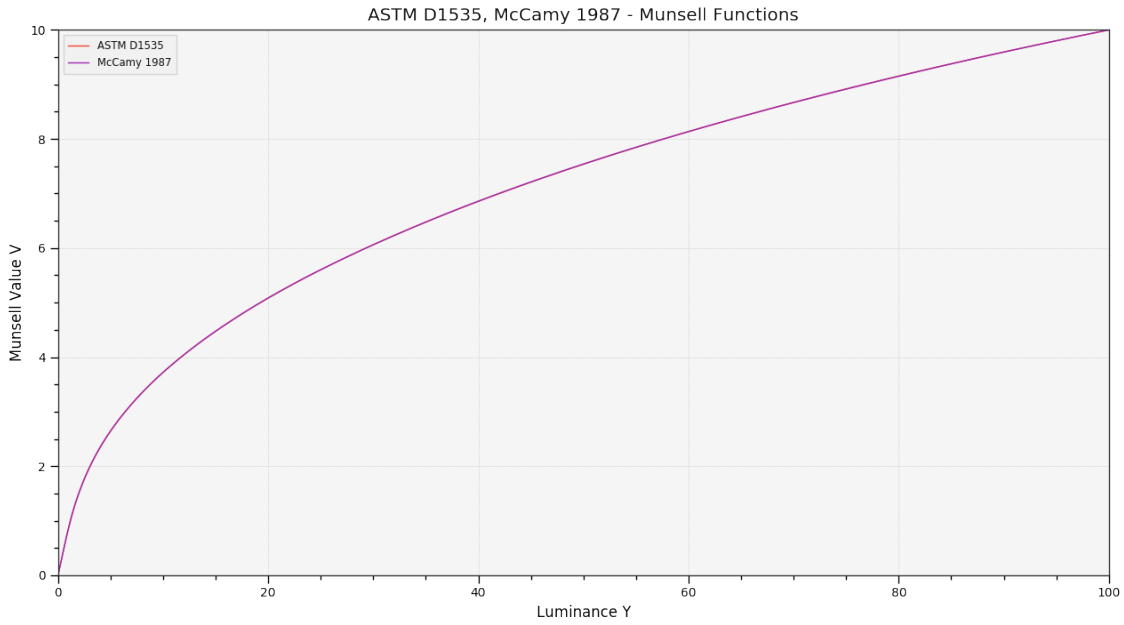
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_multi_functions(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_multi_munsell_value_functions(['ASTM D1535', 'McCamy 1987'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Optical Phenomena

colour.plotting

plot_single_sd_rayleigh_scattering([...])	Plots a single *Rayleigh* scattering spectral distribution.
plot_the_blue_sky([cmfs])	Plots the blue sky.

colour.plotting.plot_single_sd_rayleigh_scattering

colour.plotting.**plot_single_sd_rayleigh_scattering**(*CO2_concentration=300*, *temperature=288.15*, *pressure=101325*, *latitude=0*, *altitude=0*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots a single *Rayleigh* scattering spectral distribution.

Parameters

- **CO2_concentration** (numeric, optional) – CO_2 concentration in parts per million (ppm).
- **temperature** (numeric, optional) – Air temperature $T[K]$ in kelvin degrees.
- **pressure** (numeric) – Surface pressure P of the measurement site.
- **latitude** (numeric, optional) – Latitude of the site in degrees.
- **altitude** (numeric, optional) – Altitude of the site in meters.
- **cmfs** (unicode, optional) – Standard observer colour matching functions.

Other Parameters

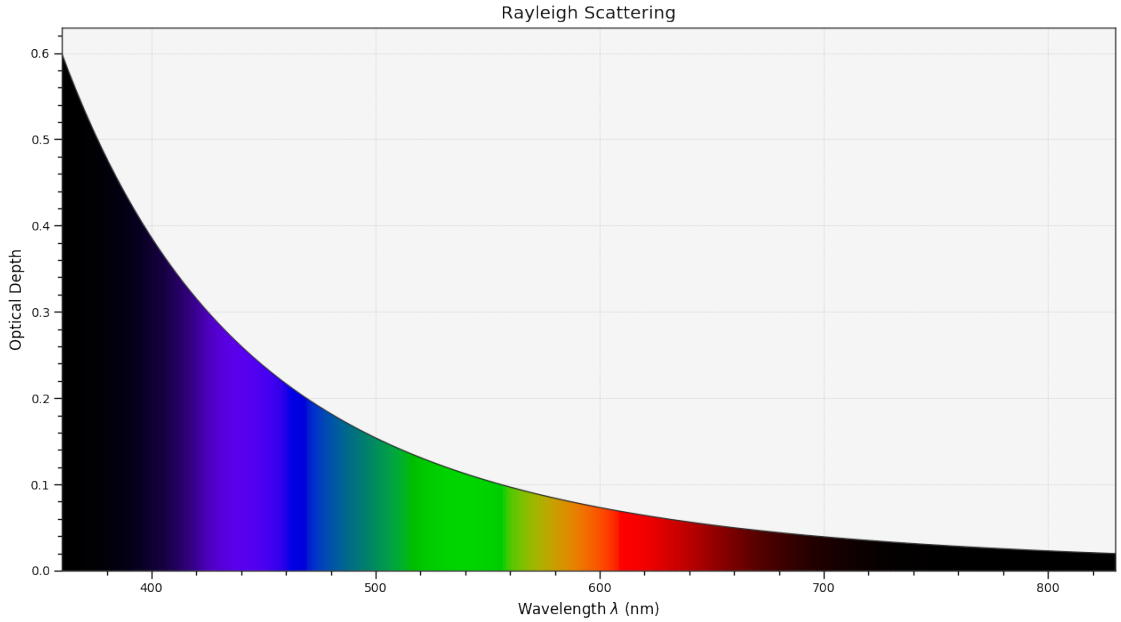
- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **out_of_gamut_clipping** (*bool, optional*) – {colour.plotting.plot_single_sd()}, Whether to clip out of gamut colours otherwise, the colours will be offset by the absolute minimal colour leading to a rendering on gray background, less saturated and smoother.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_single_sd_rayleigh_scattering()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_the_blue_sky

colour.plotting.**plot_the_blue_sky**(*cmfs='CIE 1931 2 Degree Standard Observer', **kwargs*)

Plots the blue sky.

Parameters **cmfs** (unicode, optional) – Standard observer colour matching functions.

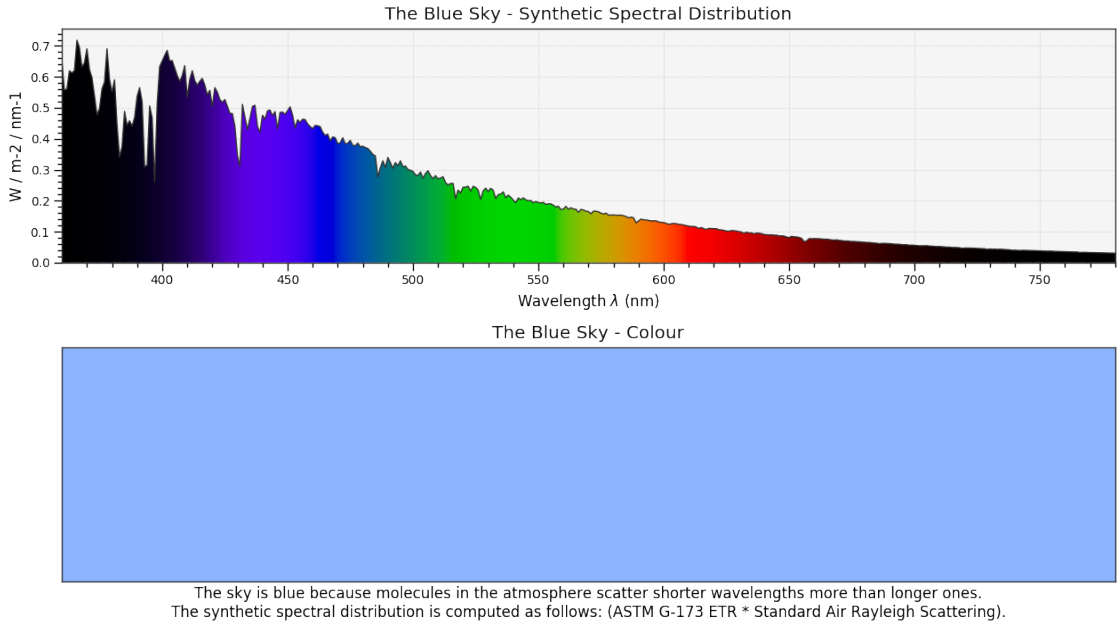
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_single_sd(), colour.plotting.plot_multi_colour_swatches(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_the_blue_sky()
(<Figure size ... with 2 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Quality

colour.plotting

plot_single_sd_colour_rendering_index_bars(sd, ...)	Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.
plot_multi_sds_colour_rendering_indexes_bars(...)	Plots the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral distributions.
plot_single_sd_colour_quality_scale_bars(sd)	Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.
plot_multi_sds_colour_quality_scales_bars(sds)	Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

colour.plotting.plot_single_sd_colour_rendering_index_bars

colour.plotting.**plot_single_sd_colour_rendering_index_bars**(*sd*, ***kwargs*)

Plots the *Colour Rendering Index* (CRI) of given illuminant or light source spectral distribution.

Parameters **sd** (SpectralDistribution) – Illuminant or light source spectral distribution to plot the *Colour Rendering Index* (CRI).

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **labels** (*bool, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.

plot_colour_quality_bars()}, Use hatching for the bars.

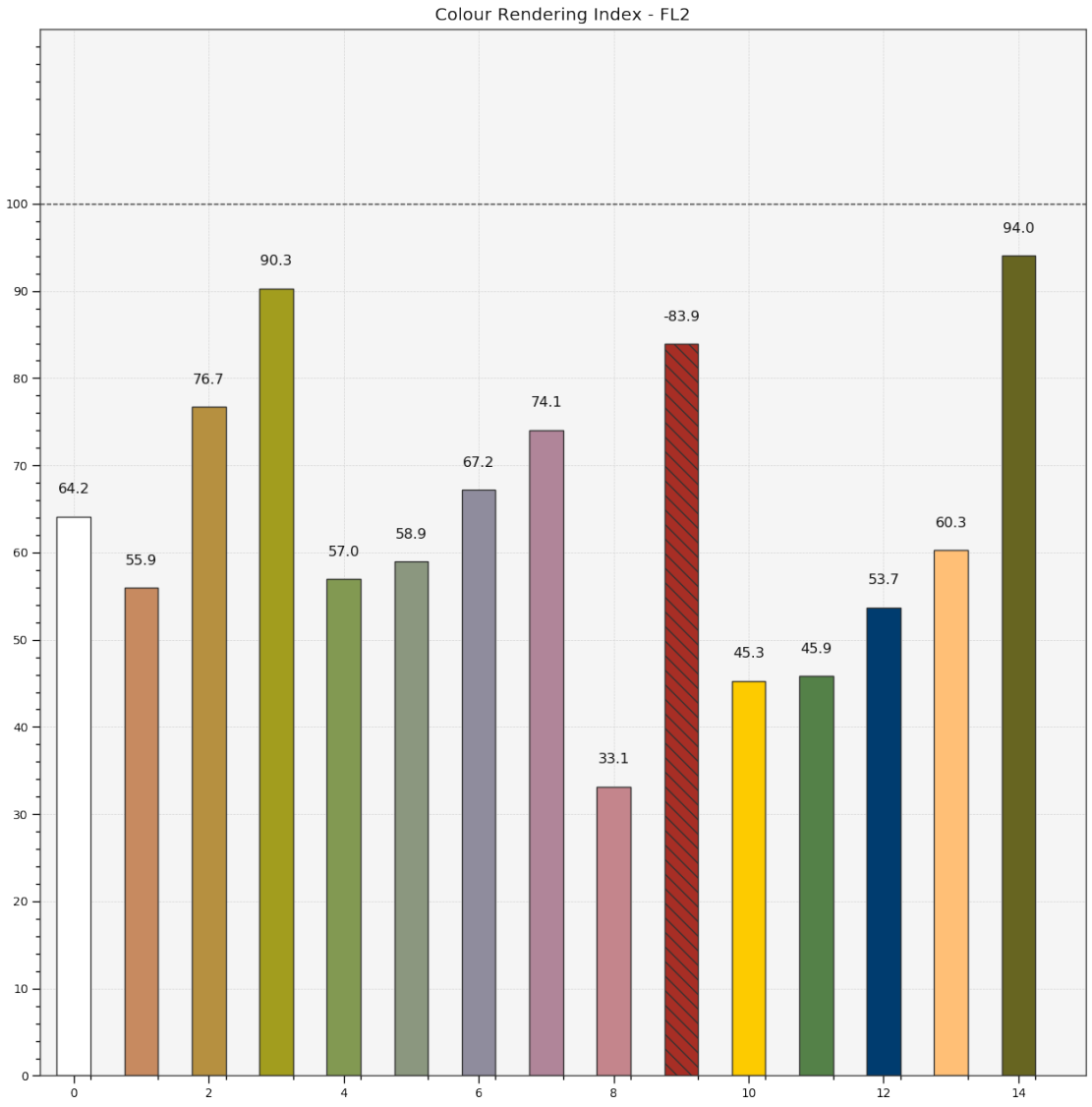
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> plot_single_sd_colour_rendering_index_bars(illuminant)
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_sds_colour_rendering_indexes_bars

colour.plotting.**plot_multi_sds_colour_rendering_indexes_bars**(*sds*, ***kwargs*)

Plots the *Colour Rendering Index* (CRI) of given illuminants or light sources spectral distributions.

Parameters **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.

Other Parameters

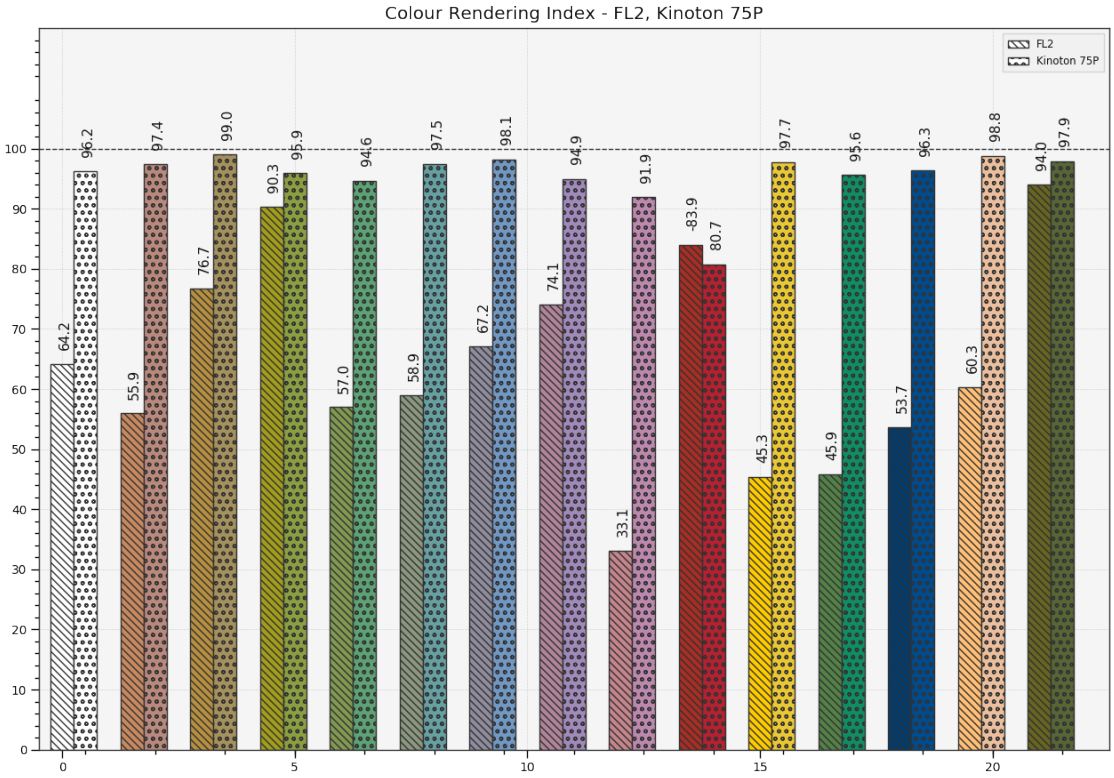
- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **labels** (*bool, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import (ILLUMINANTS_SDS,
...                     LIGHT_SOURCES_SDS)
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> light_source = LIGHT_SOURCES_SDS['Kinoton 75P']
>>> plot_multi_sds_colour_rendering_indexes_bars(
...     [illuminant, light_source])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_single_sd_colour_quality_scale_bars

colour.plotting.**plot_single_sd_colour_quality_scale_bars**(*sd*, *method='NIST CQS 9.0'*, ***kwargs*)

Plots the *Colour Quality Scale* (CQS) of given illuminant or light source spectral distribution.

Parameters

- **sd** (SpectralDistribution) – Illuminant or light source spectral distribution to plot the *Colour Quality Scale* (CQS).
- **method** (unicode, optional) – **{'NIST CQS 7.4'}**, *Colour Quality Scale* (CQS) computation method.

Other Parameters

- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **labels** (*bool, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Add labels above bars.

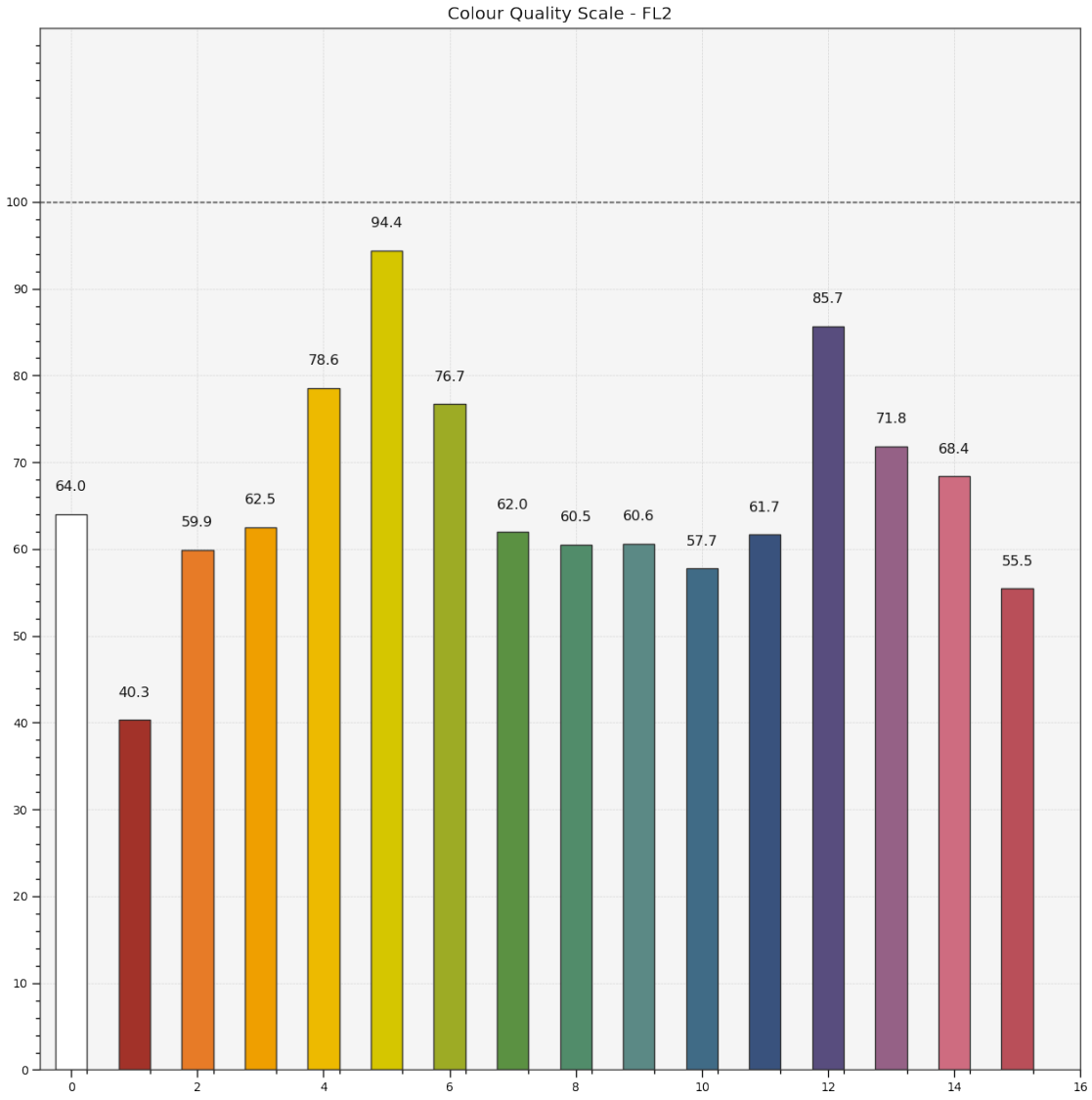
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> plot_single_sd_colour_quality_scale_bars(illuminant)
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_multi_sds_colour_quality_scales_bars

colour.plotting.**plot_multi_sds_colour_quality_scales_bars**(*sds*, *method='NIST CQS 9.0'*, ***kwargs*)

Plots the *Colour Quality Scale* (CQS) of given illuminants or light sources spectral distributions.

Parameters

- **sds** (array_like or MultiSpectralDistributions) – Spectral distributions or multi-spectral distributions to plot. *sds* can be a single colour.MultiSpectralDistributions class instance, a list of colour.MultiSpectralDistributions class instances or a list of colour.SpectralDistribution class instances.
- **method** (unicode, optional) – **{'NIST CQS 7.4'}**, *Colour Quality Scale* (CQS) computation method.

Other Parameters

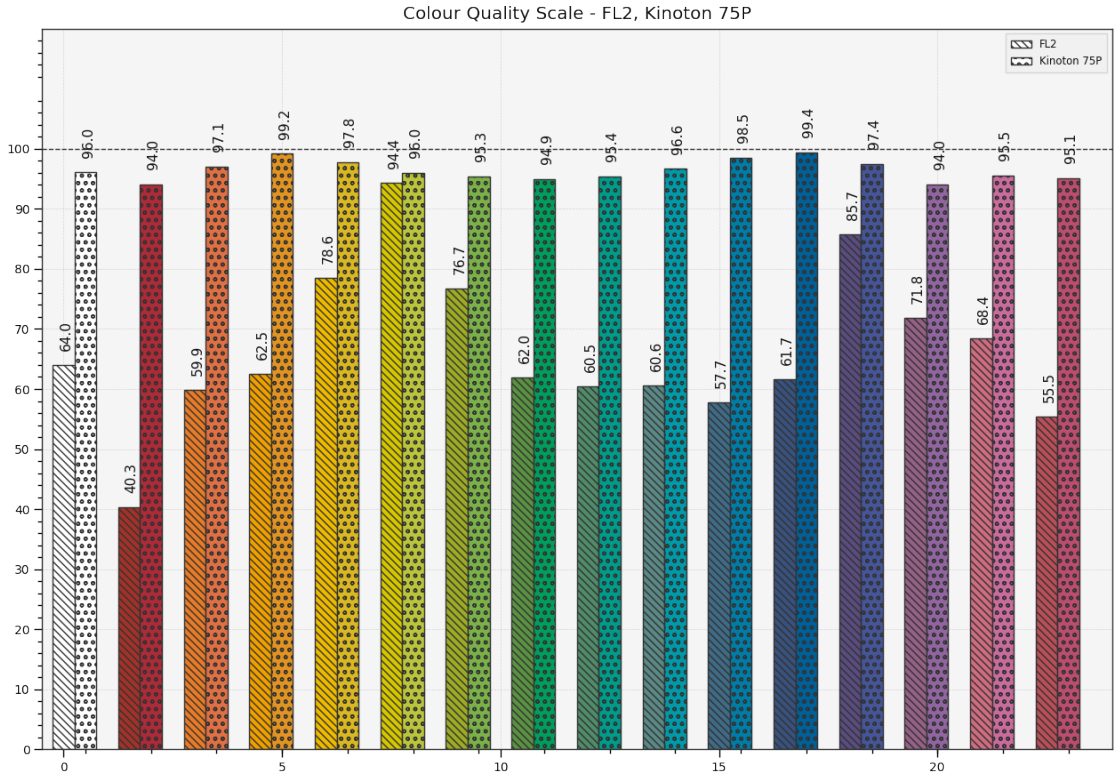
- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.
- **labels** (*bool, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Add labels above bars.
- **hatching** (*bool or None, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Use hatching for the bars.
- **hatching_repeat** (*int, optional*) – {colour.plotting.quality.plot_colour_quality_bars()}, Hatching pattern repeat.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import (ILLUMINANTS_SDS,
...                     LIGHT_SOURCES_SDS)
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> light_source = LIGHT_SOURCES_SDS['Kinoton 75P']
>>> plot_multi_sds_colour_quality_scales_bars([illuminant, light_source])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Ancillary Objects

`colour.plotting.quality`

`plot_colour_quality_bars`(specifications[, ...])	Plots the colour quality data of given illuminants or light sources colour quality specifications.

colour.plotting.quality.plot_colour_quality_bars

`colour.plotting.quality.`**`plot_colour_quality_bars`**(*specifications*, *labels=True*, *hatching=None*, *hatching_repeat=2*, ***kwargs*)

Plots the colour quality data of given illuminants or light sources colour quality specifications.

Parameters

- **`specifications`** (`array_like`) – Array of illuminants or light sources colour quality specifications.
- **`labels`** (`bool`, `optional`) – Add labels above bars.
- **`hatching`** (`bool` or `None`, `optional`) – Use hatching for the bars.
- **`hatching_repeat`** (`int`, `optional`) – Hatching pattern repeat.

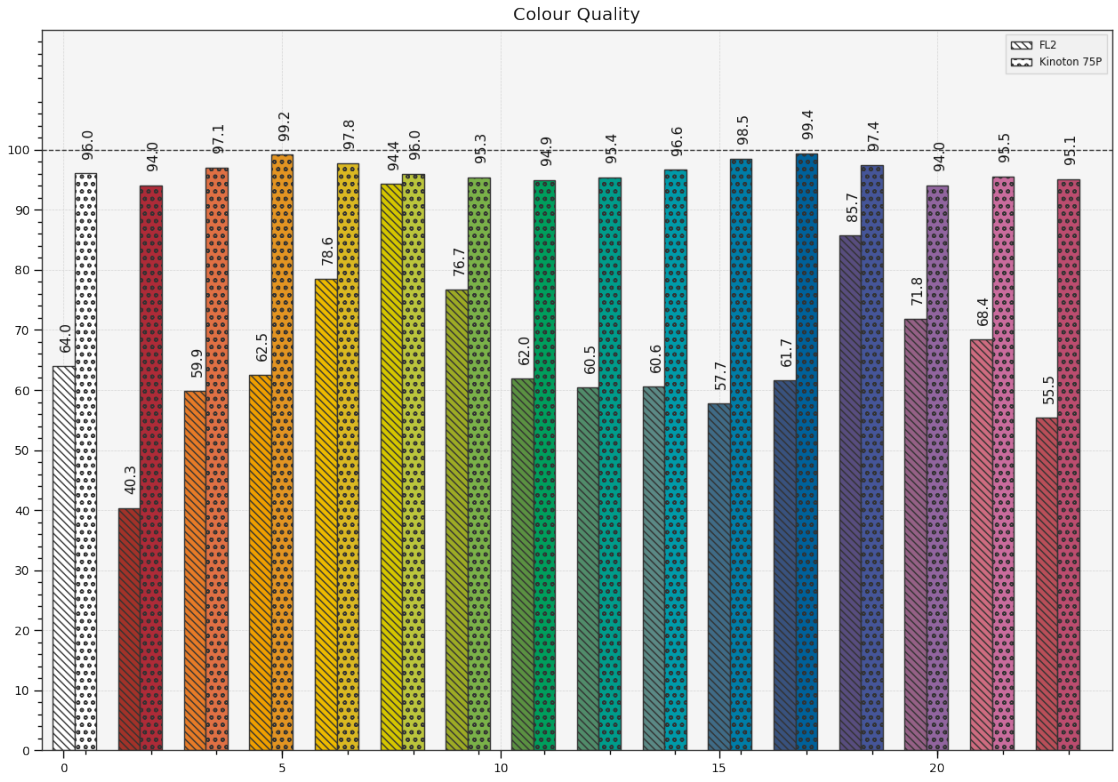
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.quality.plot_colour_quality_bars(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> from colour import (ILLUMINANTS_SDS,
...                     LIGHT_SOURCES_SDS, SpectralShape)
>>> illuminant = ILLUMINANTS_SDS['FL2']
>>> light_source = LIGHT_SOURCES_SDS['Kinoton 75P']
>>> light_source = light_source.copy().align(SpectralShape(360, 830, 1))
>>> cqs_i = colour_quality_scale(illuminant, additional_data=True)
>>> cqs_l = colour_quality_scale(light_source, additional_data=True)
>>> plot_colour_quality_bars([cqs_i, cqs_l])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Temperature & Correlated Colour Temperature

colour.plotting

plot_planckian_locus_in_chromaticity_diagram_CIE1931([...])	Plots the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.
plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS([...])	Plots the *Planckian Locus* and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.

colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1931

colour.plotting.**plot_planckian_locus_in_chromaticity_diagram_CIE1931**(*illuminants=None*, *annotate_parameters=None*, *chromaticity_diagram_callable_CIE1931=<function plot_chromaticity_diagram_CIE1931>*, *planckian_locus_callable_CIE1931=<function plot_planckian_locus_CIE1931>*, ***kwargs*)

Plots the *Planckian Locus* and given illuminants in *CIE 1931 Chromaticity Diagram*.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names if annotate is set to *True*. annotate_parameters can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each illuminant.
- **chromaticity_diagram_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *CIE 1931 Chromaticity Diagram*.
- **planckian_locus_callable_CIE1931** (callable, optional) – Callable responsible for drawing the *Planckian Locus* according to *CIE 1931* method.

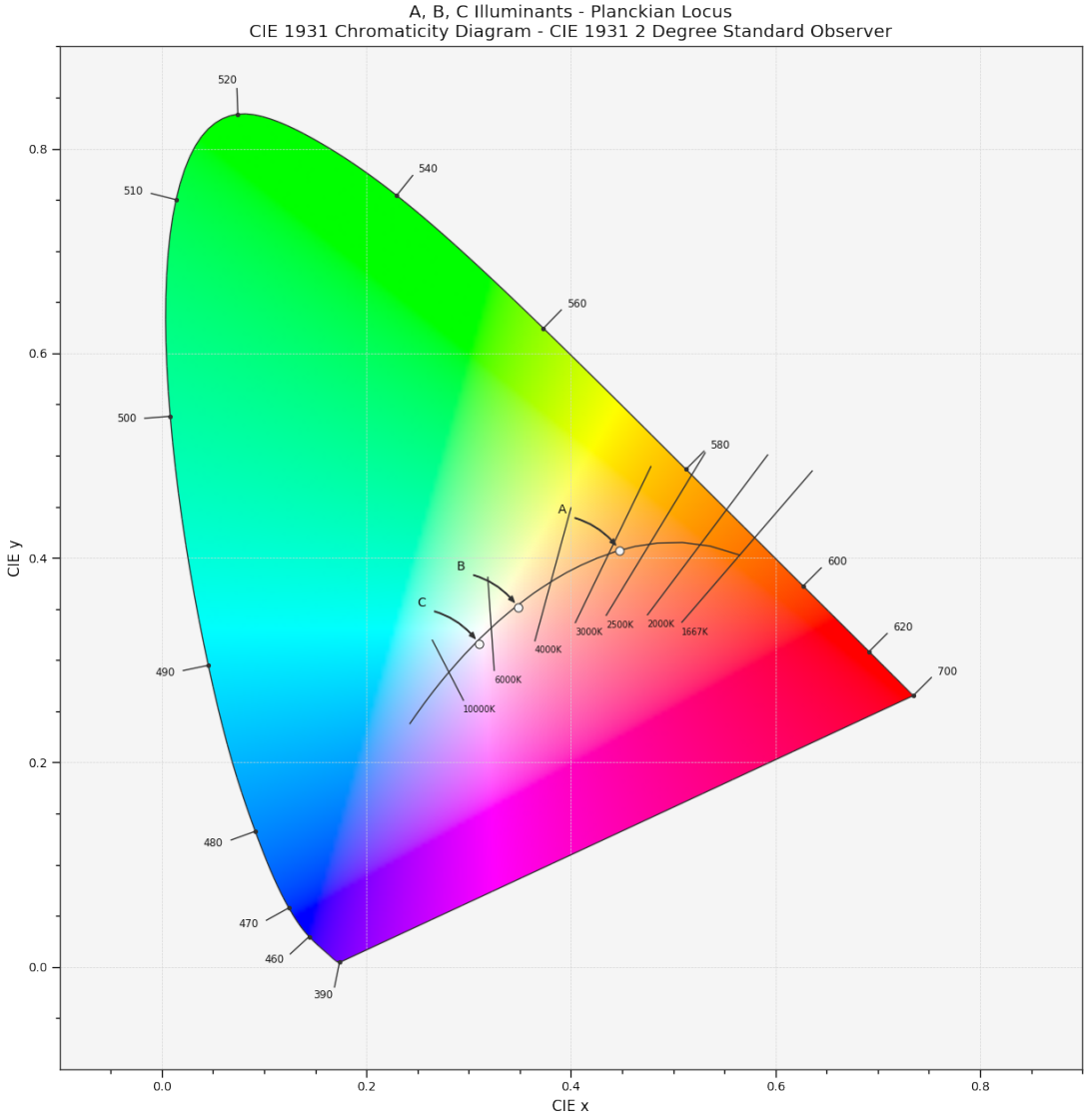
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.temperature.plot_planckian_locus(), colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1931(['A', 'B', 'C'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS

colour.plotting.**plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS**(*illuminants=None*, *annotate_parameters=None*, *chromaticity_diagram_callable_CIE1960UCS=<*, *plot_chromaticity_diagram_CIE1960UC*, *planckian_locus_callable_CIE1960UCS=<fun*, *plot_planckian_locus_CIE1960UCS>*, ***kwargs*)

Plots the *Planckian Locus* and given illuminants in *CIE 1960 UCS Chromaticity Diagram*.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names if annotate is set to *True*. annotate_parameters can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each illuminant.
- **chromaticity_diagram_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *CIE 1960 UCS Chromaticity Diagram*.
- **planckian_locus_callable_CIE1960UCS** (callable, optional) – Callable responsible for drawing the *Planckian Locus* according to *CIE 1960 UCS* method.

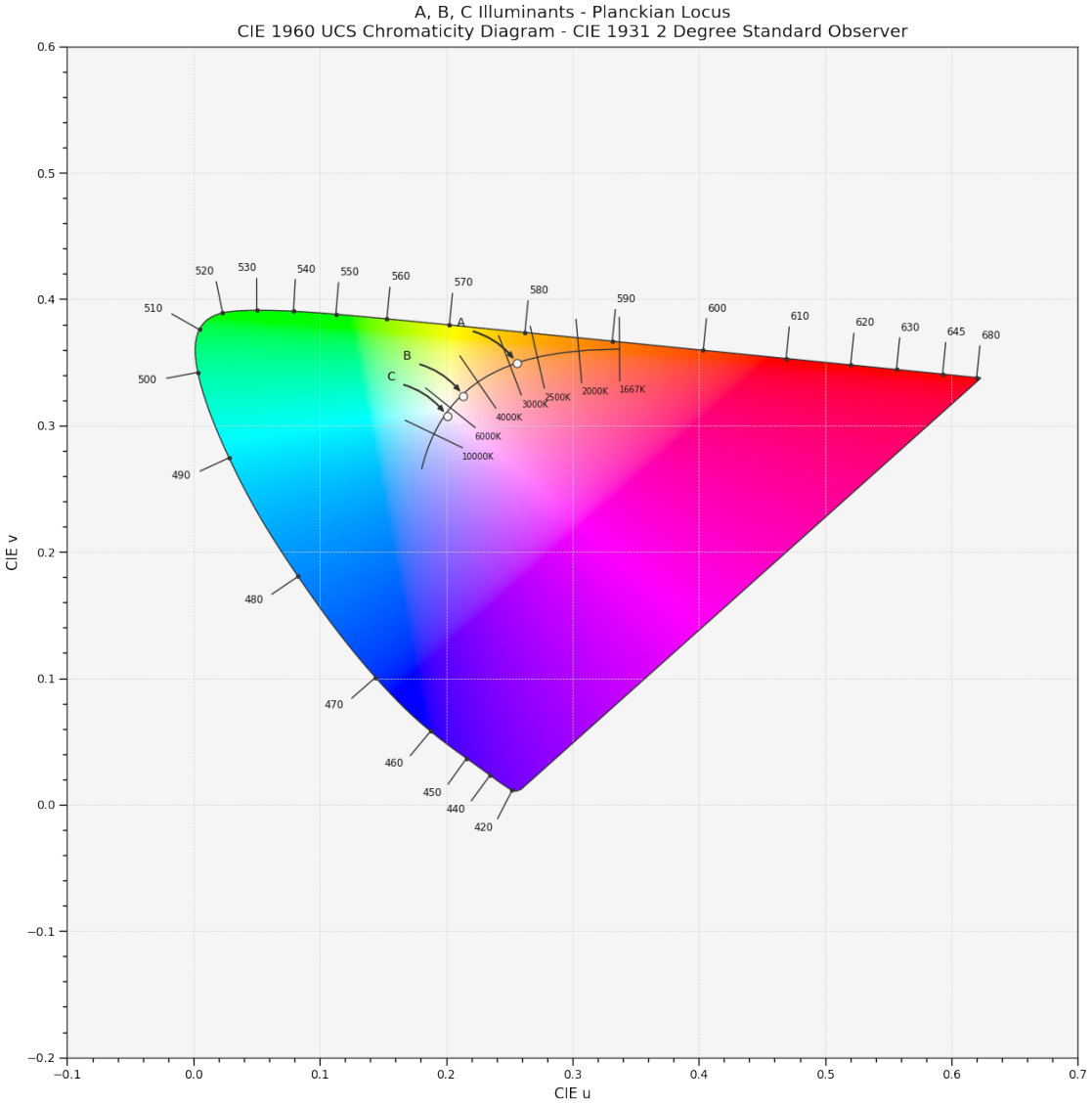
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.temperature.plot_planckian_locus(), colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(
...     ['A', 'C', 'E'])
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Ancillary Objects

`colour.plotting.temperature`

`plot_planckian_locus`([...])	Plots the *Planckian Locus* according to given method.
`plot_planckian_locus_CIE1931`([...])	Plots the *Planckian Locus* according to *CIE 1931* method.
`plot_planckian_locus_CIE1960UCS`([...])	Plots the *Planckian Locus* according to *CIE 1960 UCS* method.
`plot_planckian_locus_in_chromaticity_diagram`([...])	Plots the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given method.

colour.plotting.temperature.plot_planckian_locus

colour.plotting.temperature.**plot_planckian_locus**(*planckian_locus_colours=None*, *method='CIE 1931'*, ***kwargs*)

Plots the *Planckian Locus* according to given method.

Parameters

- **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.

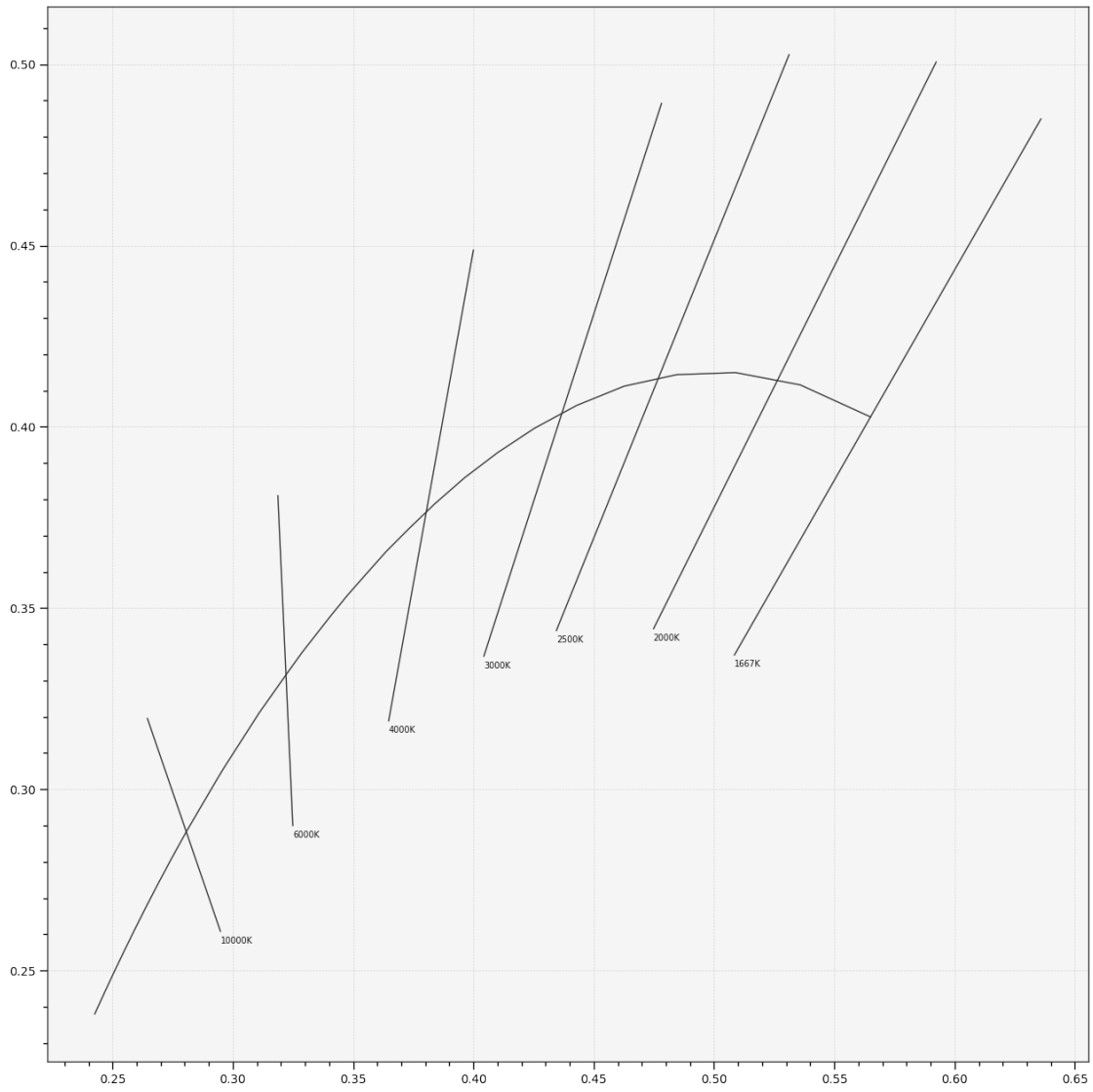
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.temperature.plot_planckian_locus_CIE1931

colour.plotting.temperature.**plot_planckian_locus_CIE1931**(*planckian_locus_colours=None*, ***kwargs*)

Plots the *Planckian Locus* according to *CIE 1931* method.

Parameters **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.

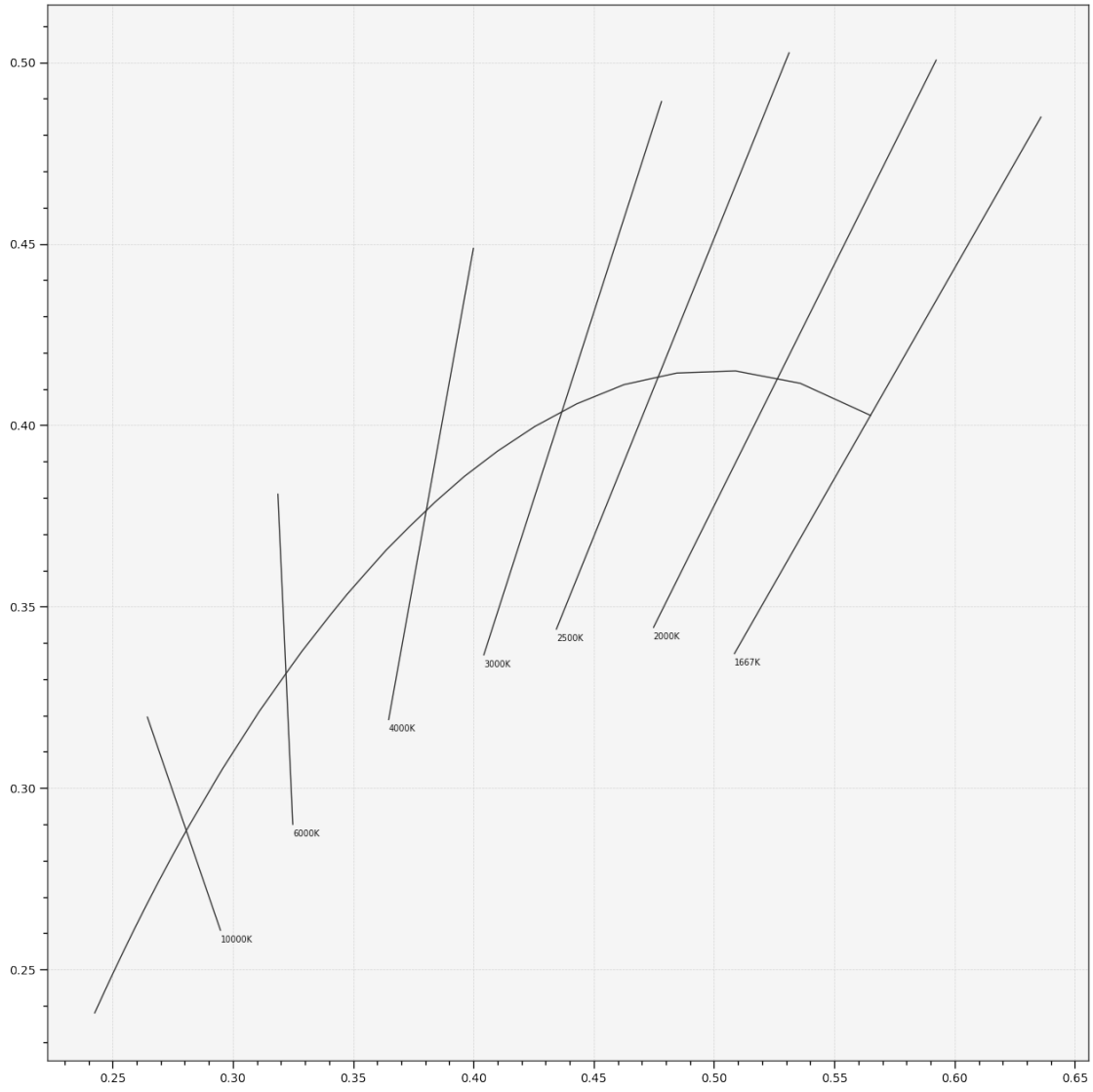
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_CIE1931()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.temperature.plot_planckian_locus_CIE1960UCS

colour.plotting.temperature.**plot_planckian_locus_CIE1960UCS**(*planckian_locus_colours=None*, ***kwargs*)

Plots the *Planckian Locus* according to *CIE 1960 UCS* method.

Parameters **planckian_locus_colours** (array_like or unicode, optional) – *Planckian Locus* colours.

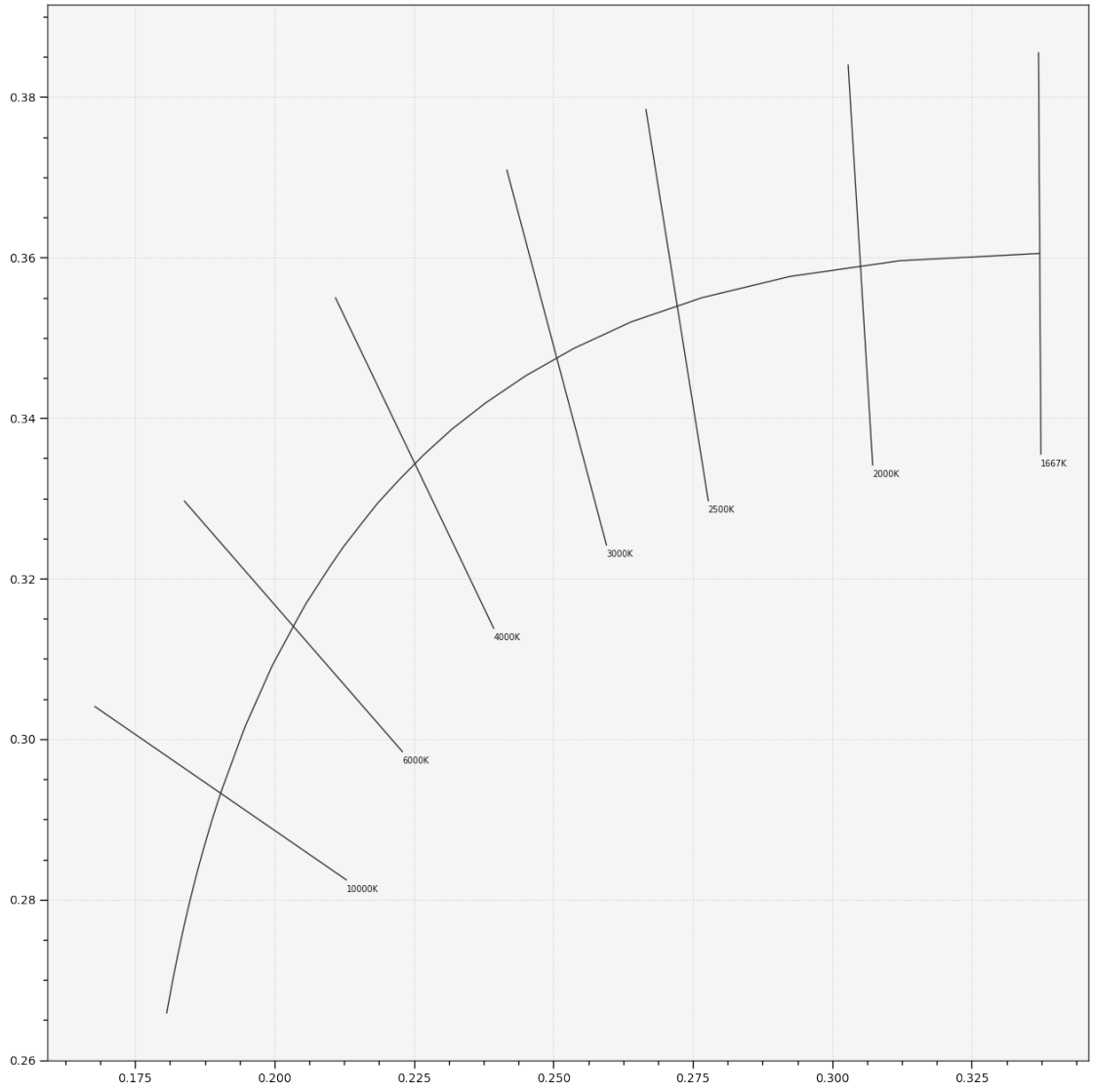
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_CIE1960UCS()
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



colour.plotting.temperature.plot_planckian_locus_in_chromaticity_diagram

colour.plotting.temperature.**plot_planckian_locus_in_chromaticity_diagram**(*illuminants=None*, *annotate_parameters=None*, *chromaticity_diagram_callable=<function plot_chromaticity_diagram>*, *planckian_locus_callable=<function plot_planckian_locus>*, *method='CIE 1931'*, ***kwargs*)

Plots the *Planckian Locus* and given illuminants in the *Chromaticity Diagram* according to given

method.

Parameters

- **illuminants** (array_like, optional) – Factory illuminants to plot.
- **annotate_parameters** (dict or array_like, optional) – Parameters for the plt.annotate() definition, used to annotate the resulting chromaticity coordinates with their respective illuminant names if annotate is set to *True*. annotate_parameters can be either a single dictionary applied to all the arrows with same settings or a sequence of dictionaries with different settings for each illuminant.
- **chromaticity_diagram_callable** (callable, optional) – Callable responsible for drawing the *Chromaticity Diagram*.
- **planckian_locus_callable** (callable, optional) – Callable responsible for drawing the *Planckian Locus*.
- **method** (unicode, optional) – **{'CIE 1931', 'CIE 1960 UCS', 'CIE 1976 UCS'}**, *Chromaticity Diagram* method.

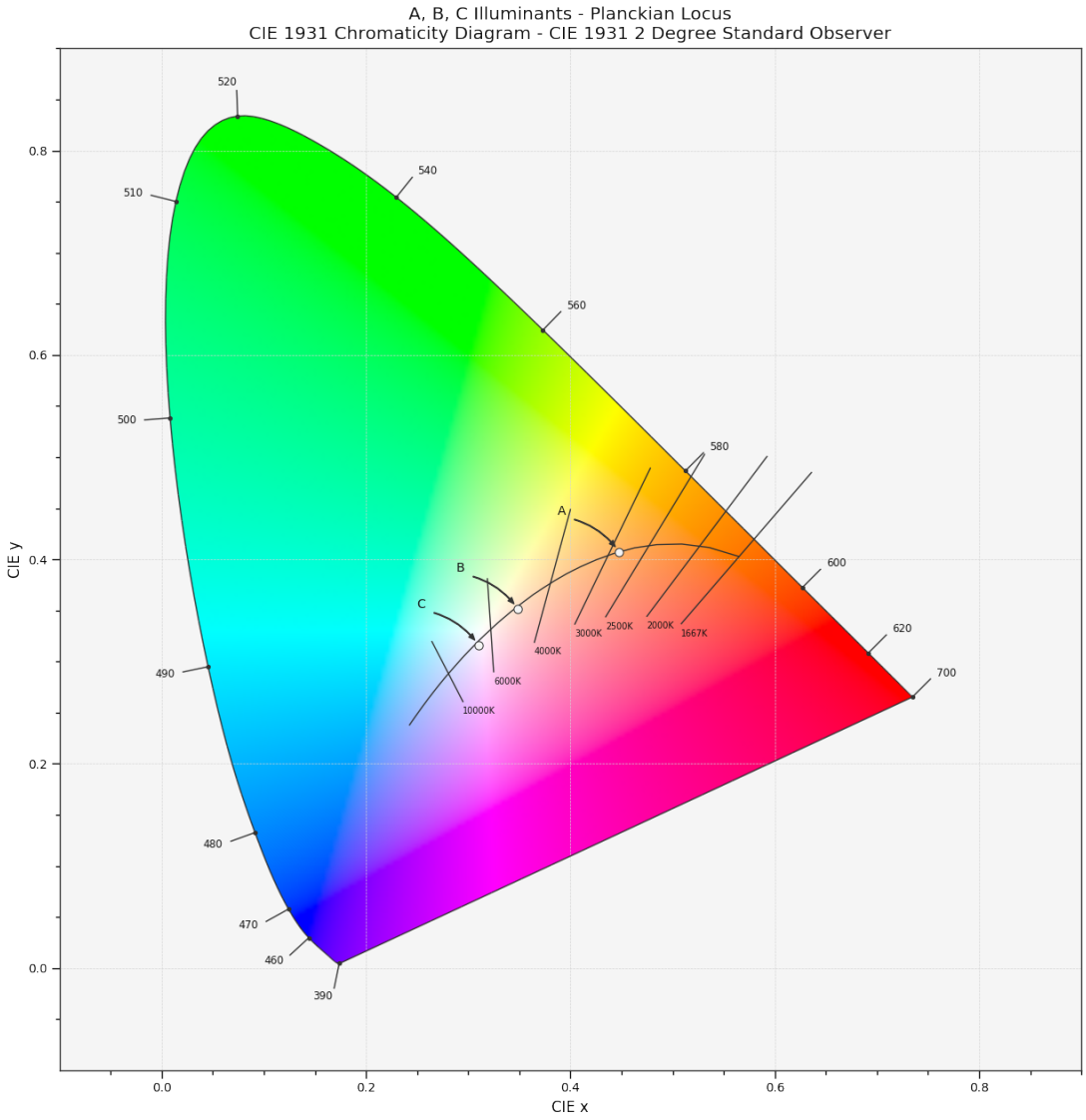
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.diagrams.plot_chromaticity_diagram(), colour.plotting.temperature.plot_planckian_locus(), colour.plotting.render()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_planckian_locus_in_chromaticity_diagram(['A', 'B', 'C'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.AxesSubplot object at 0x...>)
```



Colour Models Volume

`colour.plotting`

`plot_RGB_colourspaces_gamuts`([colourspaces, ...])	Plots given *RGB* colourspaces gamuts in given reference colourspace.
`plot_RGB_scatter`(RGB, colourspace[, ...])	Plots given *RGB* colourspace array in a scatter plot.

colour.plotting.plot_RGB_colourspaces_gamuts

colour.plotting.**plot_RGB_colourspaces_gamuts**(*colourspaces=None*, *reference_colourspace='CIE xyY'*, *segments=8*, *show_grid=True*, *grid_segments=10*, *show_spectral_locus=False*, *spectral_locus_colour=None*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots given *RGB* colourspaces gamuts in given reference colourspace.

Parameters

- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot the gamuts.
- **reference_colourspace** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE xy', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE Luv uv', 'CIE LCHuv', 'CIE UCS', 'CIE UCS uv', 'CIE UVW', 'DIN 99', 'Hunter Lab', 'Hunter Rdab', 'IPT', 'JzAzBz', 'OSA UCS', 'hdr-CIELAB', 'hdr-IPT'}**, Reference colourspace to plot the gamuts into.
- **segments** (int, optional) – Edge segments count for each *RGB* colourspace cubes.
- **show_grid** (bool, optional) – Whether to show a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **show_spectral_locus** (bool, optional) – Whether to show the spectral locus.
- **spectral_locus_colour** (array_like, optional) – Spectral locus colour.
- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectral locus.

Other Parameters

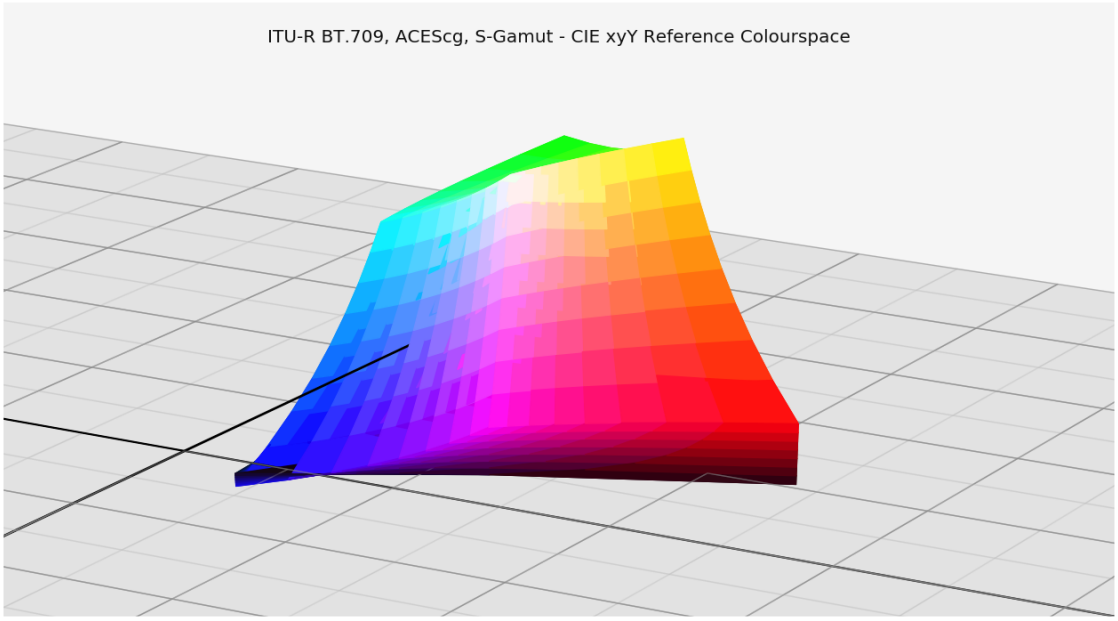
- ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.volume.nadir_grid()}, Please refer to the documentation of the previously listed definitions.
- **face_colours** (*array_like, optional*) – Face colours array such as *face_colours = (None, (0.5, 0.5, 1.0))*.
- **edge_colours** (*array_like, optional*) – Edge colours array such as *edge_colours = (None, (0.5, 0.5, 1.0))*.
- **face_alpha** (*numeric, optional*) – Face opacity value such as *face_alpha = (0.5, 1.0)*.
- **edge_alpha** (*numeric, optional*) – Edge opacity value such as *edge_alpha = (0.0, 1.0)*.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> plot_RGB_colourspaces_gamuts(['ITU-R BT.709', 'ACEScg', 'S-Gamut'])
...
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.Axes3DSubplot object at 0x...>)
```



colour.plotting.plot_RGB_scatter

colour.plotting.**plot_RGB_scatter**(*RGB*, *colourspace*, *reference_colourspace='CIE xyY'*, *colourspaces=None*, *segments=8*, *show_grid=True*, *grid_segments=10*, *show_spectral_locus=False*, *spectral_locus_colour=None*, *points_size=12*, *cmfs='CIE 1931 2 Degree Standard Observer'*, ***kwargs*)

Plots given *RGB* colourspace array in a scatter plot.

Parameters

- **RGB** (array_like) – *RGB* colourspace array.
- **colourspace** (RGB_Colourspace) – *RGB* colourspace of the *RGB* array.
- **reference_colourspace** (unicode, optional) – **{'CIE XYZ', 'CIE xyY', 'CIE xy', 'CIE Lab', 'CIE LCHab', 'CIE Luv', 'CIE Luv uv', 'CIE LCHuv', 'CIE UCS', 'CIE UCS uv', 'CIE UVW', 'DIN 99', 'Hunter Lab', 'Hunter Rdab', 'IPT', 'JzAzBz', 'OSA UCS', 'hdr-CIELAB', 'hdr-IPT'}**, Reference colourspace for colour conversion.
- **colourspaces** (array_like, optional) – *RGB* colourspaces to plot the gamuts.
- **segments** (int, optional) – Edge segments count for each *RGB* colourspace cubes.
- **show_grid** (bool, optional) – Whether to show a grid at the bottom of the *RGB* colourspace cubes.
- **grid_segments** (bool, optional) – Edge segments count for the grid.
- **show_spectral_locus** (bool, optional) – Whether to show the spectral locus.
- **spectral_locus_colour** (array_like, optional) – Spectral locus colour.
- **points_size** (numeric, optional) – Scatter points size.

- **cmfs** (unicode, optional) – Standard observer colour matching functions used for spectral locus.

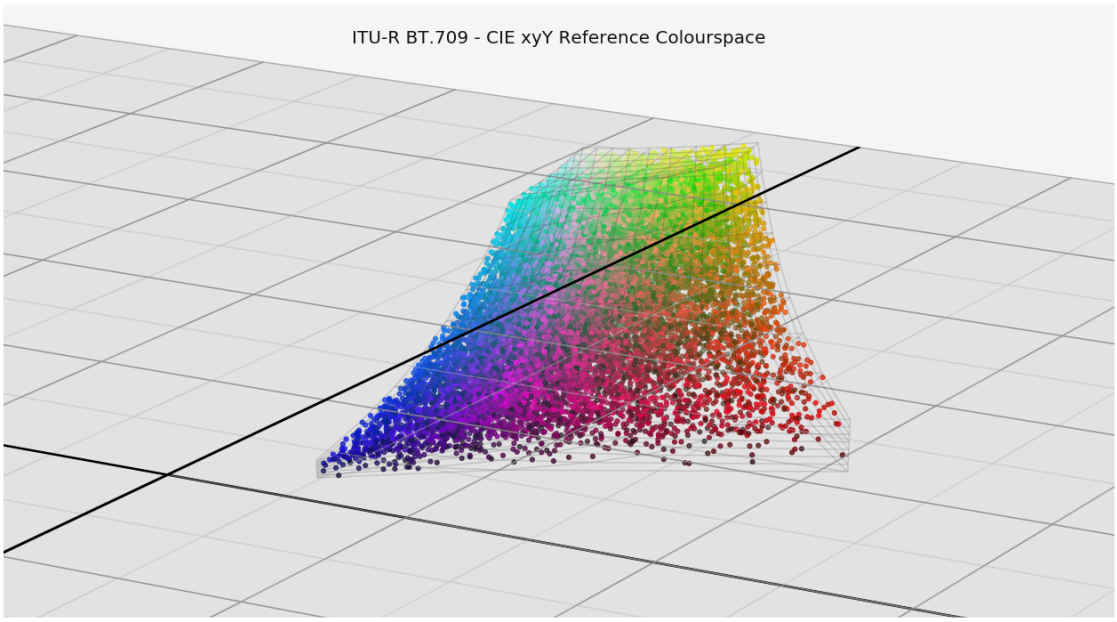
Other Parameters ****kwargs** (*dict, optional*) – {colour.plotting.artist(), colour.plotting.plot_RGB_colourspaces_gamuts()}, Please refer to the documentation of the previously listed definitions.

Returns Current figure and axes.

Return type tuple

Examples

```
>>> RGB = np.random.random((128, 128, 3))
>>> plot_RGB_scatter(RGB, 'ITU-R BT.709')
(<Figure size ... with 1 Axes>, <matplotlib.axes._subplots.Axes3DSubplot object at 0x...>)
```



Geometry Plotting Utilities

colour.plotting

quad([plane, origin, width, height, depth])	Returns the vertices of a quad geometric element in counter-clockwise order.
grid([plane, origin, width, height, depth, ...])	Returns the vertices of a grid made of quads.
cube([plane, origin, width, height, depth, ...])	Returns the vertices of a cube made of grids.

colour.plotting.quad

colour.plotting.**quad**(*plane='xy'*, *origin=None*, *width=1*, *height=1*, *depth=0*)

Returns the vertices of a quad geometric element in counter-clockwise order.

Parameters

- **plane** (array_like, optional) – **{'xy', 'xz', 'yz'}**, Construction plane of the quad.
- **origin** (array_like, optional) – Quad origin on the construction plane.
- **width** (numeric, optional) – Quad width.
- **height** (numeric, optional) – Quad height.
- **depth** (numeric, optional) – Quad depth.

Returns Quad vertices.

Return type ndarray

Examples

```
>>> quad()
array([[0, 0, 0],
       [1, 0, 0],
       [1, 1, 0],
       [0, 1, 0]])
```

colour.plotting.grid

colour.plotting.**grid**(*plane='xy'*, *origin=None*, *width=1*, *height=1*, *depth=0*, *width_segments=1*, *height_segments=1*)

Returns the vertices of a grid made of quads.

Parameters

- **plane** (array_like, optional) – **{'xy', 'xz', 'yz'}**, Construction plane of the grid.
- **origin** (array_like, optional) – Grid origin on the construction plane.
- **width** (numeric, optional) – Grid width.
- **height** (numeric, optional) – Grid height.
- **depth** (numeric, optional) – Grid depth.
- **width_segments** (int, optional) – Grid segments, quad counts along the width.
- **height_segments** (int, optional) – Grid segments, quad counts along the height.

Returns Grid vertices.

Return type ndarray

Examples

```
>>> grid(width_segments=2, height_segments=2)
array([[[ 0. ,  0. ,  0. ],
        [ 0.5,  0. ,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0. ,  0.5,  0. ]],

       [[ 0. ,  0.5,  0. ],
        [ 0.5,  0.5,  0. ],
        [ 0.5,  1. ,  0. ],
        [ 0. ,  1. ,  0. ]],

       [[ 0.5,  0. ,  0. ],
        [ 1. ,  0. ,  0. ],
        [ 1. ,  0.5,  0. ],
        [ 0.5,  0.5,  0. ]],

       [[ 0.5,  0.5,  0. ],
        [ 1. ,  0.5,  0. ],
        [ 1. ,  1. ,  0. ],
        [ 0.5,  1. ,  0. ]]])
```

colour.plotting.cube

colour.plotting.**cube**(*plane=None*, *origin=None*, *width=1*, *height=1*, *depth=1*, *width_segments=1*, *height_segments=1*, *depth_segments=1*)

Returns the vertices of a cube made of grids.

Parameters

- **plane** (array_like, optional) – Any combination of **{'+x', '-x', '+y', '-y', '+z', '-z'}**, Included grids in the cube construction.
- **origin** (array_like, optional) – Cube origin.
- **width** (numeric, optional) – Cube width.
- **height** (numeric, optional) – Cube height.
- **depth** (numeric, optional) – Cube depth.
- **width_segments** (int, optional) – Cube segments, quad counts along the width.
- **height_segments** (int, optional) – Cube segments, quad counts along the height.
- **depth_segments** (int, optional) – Cube segments, quad counts along the depth.

Returns Cube vertices.

Return type ndarray

Examples

```
>>> cube()
array([[[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 0.,  1.,  0.]],

       [[ 0.,  0.,  1.],
        [ 1.,  0.,  1.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 1.,  0.,  0.],
        [ 1.,  0.,  1.],
        [ 0.,  0.,  1.]],

       [[ 0.,  1.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 0.,  1.,  1.]],

       [[ 0.,  0.,  0.],
        [ 0.,  1.,  0.],
        [ 0.,  1.,  1.],
        [ 0.,  0.,  1.]],

       [[ 1.,  0.,  0.],
        [ 1.,  1.,  0.],
        [ 1.,  1.,  1.],
        [ 1.,  0.,  1.]]])
```

Automatic Colour Conversion Graph

`colour.plotting`

`plot_automatic_colour_conversion_graph`(filename)	Plots *Colour* automatic colour conversion graph using Graphviz and pyraphviz.

colour.plotting.plot_automatic_colour_conversion_graph

`colour.plotting.`**`plot_automatic_colour_conversion_graph`**(*filename*, *prog='fdp'*, *args=''*)

Plots *Colour* automatic colour conversion graph using Graphviz and pyraphviz.

Parameters

- **`filename`** (`unicode`) – Filename to use to save the image.
- **`prog`** (`unicode, optional`) – {'neato', 'dot', 'twopi', 'circo', 'fdp', 'nop'}, *Graphviz* layout method.
- **`args`** (`unicode, optional`) – Additional arguments for *Graphviz*.

Returns *Pyraphviz* graph.

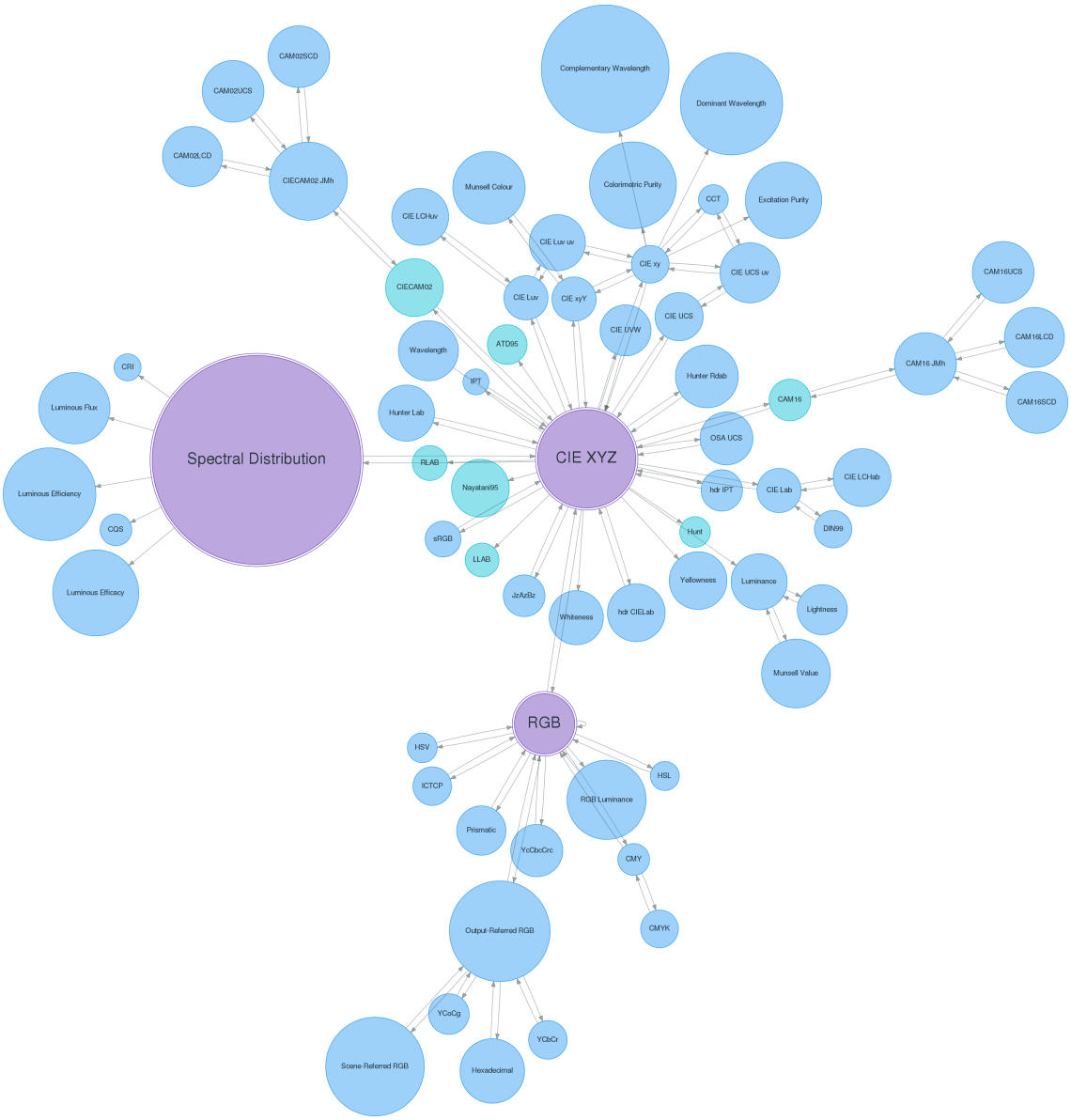
Return type AGraph

Notes

- This definition does not directly plot the *Colour* automatic colour conversion graph but instead write it to an image.

Examples

```
>>> import tempfile
>>> import colour
>>> from colour import read_image
>>> from colour.plotting import plot_image
>>> filename = '{0}.png'.format(tempfile.mkstemp()[-1])
>>> _ = plot_automatic_colour_conversion_graph(filename, 'dot')
...
>>> plot_image(read_image(filename))
```



Colour Quality

- *Colour Rendering Index*
- *Colour Quality Scale*
- *Academy Spectral Similarity Index (SSI)*

Colour Rendering Index

colour

colour_rendering_index(sd_test[, ...])	Returns the *Colour Rendering Index* (CRI) Q_a of given spectral distribution.

colour.colour_rendering_index

colour.**colour_rendering_index**(*sd_test*, *additional_data=False*)

Returns the *Colour Rendering Index* (CRI) Q_a of given spectral distribution.

Parameters

- **sd_test** (SpectralDistribution) – Test spectral distribution.
- **additional_data** (bool, optional) – Whether to output additional data.

Returns *Colour Rendering Index* (CRI).

Return type numeric or *CRI_Specification*

References

[OD08]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> sd = ILLUMINANTS_SDS['FL2']
>>> colour_rendering_index(sd)
64.1515202...
```

colour.quality

CRI_Specification	Defines the *Colour Rendering Index* (CRI) colour quality specification.

colour.quality.CRI_Specification

class colour.quality.**CRI_Specification**

Defines the *Colour Rendering Index* (CRI) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral distribution.
- **Q_a** (numeric) – *Colour Rendering Index* (CRI) Q_a.
- **Q_as** (dict) – Individual *colour rendering indexes* data for each sample.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.

References

[OD08]

Create new instance of CRI_Specification(name, Q_a, Q_as, colorimetry_data)

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

Q_a	Alias for field number 1
Q_as	Alias for field number 2
colorimetry_data	Alias for field number 3
name	Alias for field number 0

Colour Quality Scale

colour

COLOUR_QUALITY_SCALE_METHODS	Supported *Colour Quality Scale* (CQS) computation methods.
colour_quality_scale(sd_test[, ...])	Returns the *Colour Quality Scale* (CQS) of given spectral distribution using given method.

colour.COLOUR_QUALITY_SCALE_METHODS

colour.**COLOUR_QUALITY_SCALE_METHODS = ('NIST CQS 7.4', 'NIST CQS 9.0')**
Supported *Colour Quality Scale* (CQS) computation methods.

References

[DO10], [OD08], [OD13]

COLOUR_QUALITY_SCALE_METHODS [tuple] **{ 'NIST CQS 9.0', 'NIST CQS 7.4'}**

colour.colour_quality_scale

colour.**colour_quality_scale**(*sd_test*, *additional_data=False*, *method='NIST CQS 9.0'*)
Returns the *Colour Quality Scale* (CQS) of given spectral distribution using given method.

Parameters

- **sd_test** (SpectralDistribution) – Test spectral distribution.
- **additional_data** (bool, optional) – Whether to output additional data.
- **method** (unicode, optional) – **{'NIST CQS 9.0', 'NIST CQS 7.4'}**, Computation method.

Returns Color quality scale.

Return type numeric or *CQS_Specification*

References

[DO10], [OD08], [OD13]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> sd = ILLUMINANTS_SDS['FL2']
>>> colour_quality_scale(sd)
64.0172835...
```

colour.quality

CQS_Specification	Defines the *Colour Quality Scale* (CQS) colour quality specification.

colour.quality.CQS_Specification

class colour.quality.**CQS_Specification**
Defines the *Colour Quality Scale* (CQS) colour quality specification.

Parameters

- **name** (unicode) – Name of the test spectral distribution.
- **Q_a** (numeric) – Colour quality scale Q_a.
- **Q_f** (numeric) – Colour fidelity scale Q_f intended to evaluate the fidelity of object colour appearances (compared to the reference illuminant of the same correlated

colour temperature and illuminance).

- **Q_p** (numeric) – Colour preference scale Q_p similar to colour quality scale Q_a but placing additional weight on preference of object colour appearance, set to *None* in *NIST CQS 9.0* method. This metric is based on the notion that increases in chroma are generally preferred and should be rewarded.
- **Q_g** (numeric) – Gamut area scale Q_g representing the relative gamut formed by the (a^*, b^*) coordinates of the 15 samples illuminated by the test light source in the *CIE L*a*b** object colourspace.
- **Q_d** (numeric) – Relative gamut area scale Q_d, set to *None* in *NIST CQS 9.0* method.
- **Q_as** (dict) – Individual *Colour Quality Scale* (CQS) data for each sample.
- **colorimetry_data** (tuple) – Colorimetry data for the test and reference computations.

References

[DO10], [OD08], [OD13]

Create new instance of CQS_Specification(name, Q_a, Q_f, Q_p, Q_g, Q_d, Q_as, colorimetry_data)

__init__()
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.
count(value)	
index(value, [start, [stop]])	Raises ValueError if the value is not present.

Attributes

Q_a	Alias for field number 1
Q_as	Alias for field number 6
Q_d	Alias for field number 5
Q_f	Alias for field number 2
Q_g	Alias for field number 4
Q_p	Alias for field number 3
colorimetry_data	Alias for field number 7
name	Alias for field number 0

Academy Spectral Similarity Index (SSI)

colour

spectral_similarity_index(sd_test, sd_reference)	Returns the *Academy Spectral Similarity Index* (SSI) of given test spectral distribution with given reference spectral distribution.

colour.spectral_similarity_index

colour.**spectral_similarity_index**(*sd_test*, *sd_reference*)

Returns the *Academy Spectral Similarity Index* (SSI) of given test spectral distribution with given reference spectral distribution.

Parameters

- **sd_test** (SpectralDistribution) – Test spectral distribution.
- **sd_reference** (SpectralDistribution) – Reference spectral distribution.

Returns *Academy Spectral Similarity Index* (SSI).

Return type numeric

References

[TheAoMPAaSciences19]

Examples

```
>>> from colour import ILLUMINANTS_SDS
>>> sd_test = ILLUMINANTS_SDS['C']
>>> sd_reference = ILLUMINANTS_SDS['D65']
>>> spectral_similarity_index(sd_test, sd_reference)
94.0
```

Reflectance Recovery

- *CIE XYZ Colourspace to Spectral*
- *Smits (1999)*
- *Meng, Simon and Hanika (2015)*

CIE XYZ Colourspace to Spectral

colour

XYZ_to_sd(XYZ[, method])	Recovers the spectral distribution of given *CIE XYZ* tristimulus values using given method.
XYZ_TO_SD_METHODS	Supported spectral distribution recovery methods.

colour.XYZ_to_sd

colour.**XYZ_to_sd**(*XYZ*, *method='Meng 2015'*, ***kwargs*)

Recovers the spectral distribution of given *CIE XYZ* tristimulus values using given method.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **method** (unicode, optional) – **{'Meng 2015', 'Smits 1999'}**, Computation method.

Other Parameters

- **cmfs** (*XYZ_ColourMatchingFunctions*) – {colour.recovery.XYZ_to_sd_Meng2015()}, Standard observer colour matching functions.
- **interval** (*numeric, optional*) – {colour.recovery.XYZ_to_sd_Meng2015()}, Wavelength λ_i range interval in nm. The smaller interval is, the longer the computations will be.
- **optimisation_parameters** (*dict_like, optional*) – {colour.recovery.XYZ_to_sd_Meng2015()}, Parameters for scipy.optimize.minimize() definition.

Returns Recovered spectral distribution.

Return type *SpectralDistribution*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- *Smits (1999)* method will internally convert given *CIE XYZ* tristimulus values to *RGB* colourspace array assuming equal energy illuminant *E*.

References

[MSHD15], [Smi99]

Examples

Meng (2015) reflectance recovery:

```
>>> import numpy as np
>>> from colour.utilities import numpy_print_options
>>> from colour.colorimetry import (
...     STANDARD_OBSERVERS_CMFS, SpectralShape, sd_to_XYZ_integration)
>>> XYZ = np.array([0.21781186, 0.12541048, 0.04697113])
>>> cmfs = (
...     STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> sd = XYZ_to_sd(XYZ, cmfs=cmfs)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.0780114...],
                      [ 370.        ,    0.0780316...],
                      [ 380.        ,    0.0780471...],
                      [ 390.        ,    0.0780351...],
                      [ 400.        ,    0.0779702...],
                      [ 410.        ,    0.0778033...],
                      [ 420.        ,    0.0770958...],
                      [ 430.        ,    0.0748008...],
                      [ 440.        ,    0.0693230...],
                      [ 450.        ,    0.0601136...],
                      [ 460.        ,    0.0477407...],
                      [ 470.        ,    0.0334964...],
                      [ 480.        ,    0.0193352...],
                      [ 490.        ,    0.0074858...],
                      [ 500.        ,    0.0001225...],
                      [ 510.        ,    0.       ...],
                      [ 520.        ,    0.       ...],
                      [ 530.        ,    0.       ...],
                      [ 540.        ,    0.0124896...],
                      [ 550.        ,    0.0389831...],
                      [ 560.        ,    0.0775105...],
                      [ 570.        ,    0.1247947...],
                      [ 580.        ,    0.1765339...],
                      [ 590.        ,    0.2281918...],
                      [ 600.        ,    0.2751347...],
                      [ 610.        ,    0.3140115...],
                      [ 620.        ,    0.3433561...],
                      [ 630.        ,    0.3635777...],
                      [ 640.        ,    0.3765428...],
                      [ 650.        ,    0.3841726...],
                      [ 660.        ,    0.3883633...],
                      [ 670.        ,    0.3905415...],
                      [ 680.        ,    0.3916742...],
                      [ 690.        ,    0.3922554...],
                      [ 700.        ,    0.3925427...],
                      [ 710.        ,    0.3926783...],
                      [ 720.        ,    0.3927330...],
                      [ 730.        ,    0.3927586...],
                      [ 740.        ,    0.3927548...],
                      [ 750.        ,    0.3927681...],
                      [ 760.        ,    0.3927813...],
                      [ 770.        ,    0.3927840...],
                      [ 780.        ,    0.3927536...]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
```

(continues on next page)

(continued from previous page)

```
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
>>> sd_to_XYZ_integration(sd) / 100
array([ 0.2178545...,  0.1254141...,  0.0470095...])
```

Smits (1999) reflectance recovery:

```
>>> sd = XYZ_to_sd(XYZ, method='Smits 1999')
>>> with numpy_print_options(suppress=True):
...     sd
SpectralDistribution([[ 380.        ,    0.07691923],
                      [ 417.7778    ,    0.0587005 ],
                      [ 455.5556    ,    0.03943195],
                      [ 493.3333    ,    0.03024978],
                      [ 531.1111    ,    0.02750692],
                      [ 568.8889    ,    0.02808645],
                      [ 606.6667    ,    0.34298985],
                      [ 644.4444    ,    0.41185795],
                      [ 682.2222    ,    0.41185795],
                      [ 720.        ,    0.41180754]],
                     interpolator=LinearInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
>>> sd_to_XYZ_integration(sd) / 100
array([ 0.2004523...,  0.1105627...,  0.0420964...])
```

colour.XYZ_TO_SD_METHODS

colour.**XYZ_TO_SD_METHODS = CaseInsensitiveMapping({'Meng 2015': ..., 'Smits 1999': ...})**

Supported spectral distribution recovery methods.

References

[MSHD15], [Smi99]

XYZ_TO_SD_METHODS [CaseInsensitiveMapping] **{'Meng 2015', 'Smits 1999'}**

Smits (1999)

colour.recovery

RGB_to_sd_Smits1999(RGB)	Recovers the spectral distribution of given *RGB* colourspace array using *Smits (1999)* method.
SMITS_1999_SDS	*Smits (1999)* spectral distributions.

colour.recovery.RGB_to_sd_Smits1999

colour.recovery.**RGB_to_sd_Smits1999**(*RGB*)

Recovers the spectral distribution of given *RGB* colourspace array using *Smits (1999)* method.

Parameters **RGB** (array_like, (3,)) – *RGB* colourspace array to recover the spectral distribution from.

Returns Recovered spectral distribution.

Return type *SpectralDistribution*

Notes

Domain	Scale - Reference	Scale - 1
RGB	[0, 1]	[0, 1]

References

[Smi99]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> RGB = np.array([0.40639599, 0.02752894, 0.03982193])
>>> with numpy_print_options(suppress=True):
...     RGB_to_sd_Smits1999(RGB)
SpectralDistribution([[ 380.        ,    0.0769192...],
                      [ 417.7778    ,    0.0587004...],
                      [ 455.5556    ,    0.0394319...],
                      [ 493.3333    ,    0.0302497...],
                      [ 531.1111    ,    0.0275069...],
                      [ 568.8889    ,    0.0280864...],
                      [ 606.6667    ,    0.3429898...],
                      [ 644.4444    ,    0.4118579...],
                      [ 682.2222    ,    0.4118579...],
                      [ 720.        ,    0.4118075...]],
                     interpolator=LinearInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={...})
```

colour.recovery.SMITS_1999_SDS

colour.recovery.**SMITS_1999_SDS = CaseInsensitiveMapping({'white': ..., 'cyan': ..., 'magenta': ..., 'ye**

Smits (1999) spectral distributions.

References

[Smi99]

SMITS_1999_SDS : CaseInsensitiveMapping

Meng, Simon and Hanika (2015)

colour.recovery

XYZ_to_sd_Meng2015(XYZ[, cmfs, illuminant, ...])	Recovers the spectral distribution of given *CIE XYZ* tristimulus values using *Meng et al.(2015)* method..

colour.recovery.XYZ_to_sd_Meng2015

colour.recovery.**XYZ_to_sd_Meng2015**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *optimisation_parameters=None*)

Recovers the spectral distribution of given *CIE XYZ* tristimulus values using *Meng et al. (2015)* method.

Parameters

- **XYZ** (array_like, (3,)) – *CIE XYZ* tristimulus values to recover the spectral distribution from.
- **cmfs** (XYZ_ColourMatchingFunctions) – Standard observer colour matching functions. The wavelength λ_i range interval of the colour matching functions affects directly the time the computations take. The current default interval of 5 is a good compromise between precision and time spent.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns Recovered spectral distribution.

Return type *SpectralDistribution*

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

- The definition used to convert spectrum to *CIE XYZ* tristimulus values is colour.colorimetry.spectral_to_XYZ_integration() definition because it processes any measurement interval opposed to colour.colorimetry.sd_to_XYZ_ASTME308() definition that handles only measurement interval of 1, 5, 10 or 20nm.

References

[MSHD15]

Examples

```
>>> from colour.utilities import numpy_print_options
>>> XYZ = np.array([0.20654008, 0.12197225, 0.05136952])
>>> cmfs = (
...     STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer'].
...     copy().align(SpectralShape(360, 780, 10))
... )
>>> sd = XYZ_to_sd_Meng2015(XYZ, cmfs)
>>> with numpy_print_options(suppress=True):
...     # Doctests skip for Python 2.x compatibility.
...     sd
SpectralDistribution([[ 360.        ,    0.0780114...],
                      [ 370.        ,    0.0780316...],
                      [ 380.        ,    0.0780471...],
                      [ 390.        ,    0.0780351...],
                      [ 400.        ,    0.0779702...],
                      [ 410.        ,    0.0778033...],
                      [ 420.        ,    0.0770958...],
                      [ 430.        ,    0.0748008...],
                      [ 440.        ,    0.0693230...],
                      [ 450.        ,    0.0601136...],
                      [ 460.        ,    0.0477407...],
                      [ 470.        ,    0.0334964...],
                      [ 480.        ,    0.0193352...],
                      [ 490.        ,    0.0074858...],
                      [ 500.        ,    0.0001225...],
                      [ 510.        ,    0.       ...],
                      [ 520.        ,    0.       ...],
                      [ 530.        ,    0.       ...],
                      [ 540.        ,    0.0124896...],
                      [ 550.        ,    0.0389831...],
                      [ 560.        ,    0.0775105...],
                      [ 570.        ,    0.1247947...],
                      [ 580.        ,    0.1765339...],
                      [ 590.        ,    0.2281918...],
                      [ 600.        ,    0.2751347...],
                      [ 610.        ,    0.3140115...],
                      [ 620.        ,    0.3433561...],
                      [ 630.        ,    0.3635777...],
                      [ 640.        ,    0.3765428...],
                      [ 650.        ,    0.3841726...],
                      [ 660.        ,    0.3883633...],
                      [ 670.        ,    0.3905415...],
                      [ 680.        ,    0.3916742...],
                      [ 690.        ,    0.3922554...],
                      [ 700.        ,    0.3925427...],
                      [ 710.        ,    0.3926783...],
                      [ 720.        ,    0.3927330...],
                      [ 730.        ,    0.3927586...],
                      [ 740.        ,    0.3927548...],
                      [ 750.        ,    0.3927681...],
                      [ 760.        ,    0.3927813...],
                      [ 770.        ,    0.3927840...],
                      [ 780.        ,    0.3927536...]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
```

(continues on next page)

(continued from previous page)

```
                    extrapolator=Extrapolator,
                    extrapolator_args={...})
>>> sd_to_XYZ_integration(sd) / 100
array([ 0.2065812...,  0.1219752...,  0.0514132...])
```

Colour Temperature

- *Correlated Colour Temperature*
  - *Robertson (1968)*
  - *Krystek (1985)*
  - *Ohno (2013)*
  - *McCamy (1992)*
  - *Hernandez-Andres, Lee and Romero (1999)*
  - *Kang, Moon, Hong, Lee, Cho and Kim (2002)*
  - *CIE Illuminant D Series*

Correlated Colour Temperature

colour

uv_to_CCT(uv[, method])	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.
UV_TO_CCT_METHODS	Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
CCT_to_uv(CCT_D_uv[, method])	Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
CCT_TO_UV_METHODS	Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.
xy_to_CCT(xy[, method])	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using given method.
XY_TO_CCT_METHODS	Supported *CIE xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.
CCT_to_xy(CCT[, method])	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.
CCT_TO_XY_METHODS	Supported correlated colour temperature T_{cp} to *CIE xy* chromaticity coordinates computation methods.

colour.uv_to_CCT

colour.**uv_to_CCT**(*uv*, *method='Ohno 2013'*, ***kwargs*)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using given method.

Parameters

- **uv** (array_like) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **method** (unicode, optional) – **{'Ohno 2013', 'Krystek 1985, 'Robertson 1968'}**, Computation method.

Other Parameters

- **cmfs** (*XYZ_ColourMatchingFunctions, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Standard observer colour matching functions.
- **start** (*numeric, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperature range start in kelvins.
- **end** (*numeric, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperature range end in kelvins.
- **count** (*int, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Temperatures count in the planckian tables.
- **iterations** (*int, optional*) – {colour.temperature.uv_to_CCT_Ohno2013()}, Number of planckian tables to generate.
- **optimisation_parameters** (*dict_like, optional*) – {colour.temperature.uv_to_CCT_Krystek1985()}, Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type ndarray

References

[AdobeSystems13a], [AdobeSystems13b], [Kry85], [Ohn14], [WS00e]

Examples

```
>>> import numpy as np
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT(uv)
array([  6.5074738...e+03,   3.2233461...e-03])
```

colour.UV_TO_CCT_METHODS

colour.**UV_TO_CCT_METHODS = CaseInsensitiveMapping({'Krystek 1985': ..., 'Ohno 2013': ..., 'Robertson 196**

Supported *CIE UCS* colourspace *uv* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[AdobeSystems13a], [AdobeSystems13b], [Ohn14], [WS00e]

UV_TO_CCT_METHODS [CaseInsensitiveMapping] **{'Ohno 2013', 'Krystek 1985, 'Robertson 1968'}**

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.CCT_to_uv

colour.**CCT_to_uv**(*CCT_D_uv*, *method='Ohno 2013'*, ***kwargs*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT_D_uv** (ndarray) – Correlated colour temperature T_{cp}, Δ_{uv}.
- **method** (unicode, optional) – **{'Ohno 2013', 'Robertson 1968', 'Krystek 1985}**, Computation method.

Other Parameters **cmfs** (*XYZ_ColourMatchingFunctions, optional*) – {colour.temperature.CCT_to_uv_Ohno2013()}, Standard observer colour matching functions.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

References

[AdobeSystems13a], [AdobeSystems13b], [Kry85], [Ohn14], [WS00e]

Examples

```
>>> import numpy as np
>>> CCT_D_uv = np.array([6507.47380460, 0.00322335])
>>> CCT_to_uv(CCT_D_uv)
array([ 0.1977999...,  0.3121999...])
```

colour.CCT_TO_UV_METHODS

colour.**CCT_TO_UV_METHODS = CaseInsensitiveMapping({'Krystek 1985': ..., 'Ohno 2013': ..., 'Robertson 196**

Supported correlated colour temperature T_{cp} to *CIE UCS* colourspace *uv* chromaticity coordinates computation methods.

References

[AdobeSystems13a], [AdobeSystems13b], [Kry85], [Ohn14], [WS00e]

CCT_TO_UV_METHODS [CaseInsensitiveMapping] **{'Ohno 2013', 'Krystek 1985, 'Robertson 1968'}**

Aliases:

- 'ohno2013': 'Ohno 2013'
- 'robertson1968': 'Robertson 1968'

colour.xy_to_CCT

colour.**xy_to_CCT**(*xy*, *method='CIE Illuminant D Series'*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using given method.

Parameters

- **xy** (array_like) – *CIE xy* chromaticity coordinates.
- **method** (unicode, optional) – **{'CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999', 'McCamy 1992'}**, Computation method.

Other Parameters **optimisation_parameters** (*dict_like, optional*) – {colour.temperature.xy_to_CCT_CIE_D(), colour.temperature.xy_to_CCT_Kang2002()}, Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type numeric or ndarray

References

[HernandezAndresLR99], [KMH+02], [Wik01a], [Wik01b], [WS00d]

Examples

```
>>> import numpy as np
>>> xy_to_CCT(np.array([0.31270, 0.32900]))
6508.1175148...
>>> xy_to_CCT(np.array([0.31270, 0.32900]), 'Hernandez 1999')
...
6500.7420431...
```

colour.XY_TO_CCT_METHODS

colour.**XY_TO_CCT_METHODS = CaseInsensitiveMapping({'CIE Illuminant D Series': ..., 'Hernandez 1999': ...**

Supported *CIE xy* chromaticity coordinates to correlated colour temperature T_{cp} computation methods.

References

[HernandezAndresLR99], [KMH+02], [Wik01a], [Wik01b], [WS00d]

XY_TO_CCT_METHODS [CaseInsensitiveMapping] **{'McCamy 1992', 'CIE Illuminant D Series', 'Kang 2002', 'Hernandez 1999'}**

Aliases:

- 'daylight': 'CIE Illuminant D Series'
- 'kang2002': 'Kang 2002'
- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

colour.CCT_to_xy

colour.**CCT_to_xy**(*CCT*, *method='CIE Illuminant D Series'*)

Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using given method.

Parameters

- **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.
- **method** (unicode, optional) – **{'CIE Illuminant D Series', 'Hernandez 1999', 'Kang 2002', 'McCamy 1992'}**, Computation method.

Other Parameters **optimisation_parameters** (*dict_like, optional*) – {colour.temperature.CCT_to_xy_Hernandez1999(), colour.temperature.CCT_to_xy_McCamy1992()}, Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

References

[HernandezAndresLR99], [KMH+02], [Wik01a], [Wik01b], [WS00d]

Examples

```
>>> CCT_to_xy(6504.38938305)
array([ 0.3127077...,  0.3291128...])
>>> CCT_to_xy(6504.38938305, 'Kang 2002')
...
array([ 0.313426 ...,  0.3235959...])
```

colour.CCT_TO_XY_METHODS

colour.**CCT_TO_XY_METHODS = CaseInsensitiveMapping({'CIE Illuminant D Series': ..., 'Hernandez 1999': ...**

Supported correlated colour temperature T_{cp} to *CIE xy* chromaticity coordinates computation methods.

References

[HernandezAndresLR99], [KMH+02], [Wik01a], [Wik01b], [WS00d]

CCT_TO_XY_METHODS [CaseInsensitiveMapping] **{'Kang 2002', 'CIE Illuminant D Series', 'Hernandez 1999', 'McCamy 1992'}**

Aliases:

- 'daylight': 'CIE Illuminant D Series'
- 'kang2002': 'Kang 2002'
- 'mccamy1992': 'McCamy 1992'
- 'hernandez1999': 'Hernandez 1999'

Robertson (1968)

`colour.temperature`

`uv_to_CCT_Robertson1968`(uv)	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Roberston (1968)* method.
`CCT_to_uv_Robertson1968`(CCT_D_uv)	Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using *Roberston (1968)* method.

colour.temperature.uv_to_CCT_Robertson1968

`colour.temperature.`**`uv_to_CCT_Robertson1968`**(*uv*)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Roberston (1968)* method.

Parameters **uv** (`array_like`) – *CIE UCS* colourspace *uv* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type ndarray

References

[AdobeSystems13a], [WS00e]

Examples

```
>>> uv = np.array([0.193741375998230, 0.315221043940594])
>>> uv_to_CCT_Robertson1968(uv)
array([  6.5000162...e+03,   8.3333289...e-03])
```

colour.temperature.CCT_to_uv_Robertson1968

colour.temperature.**CCT_to_uv_Robertson1968**(*CCT_D_uv*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} and Δ_{uv} using *Roberston (1968)* method.

Parameters **CCT_D_uv** (ndarray) – Correlated colour temperature T_{cp}, Δ_{uv}.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

References

[AdobeSystems13b], [WS00e]

Examples

```
>>> CCT_D_uv = np.array([6500.0081378199056, 0.008333331244225])
>>> CCT_to_uv_Robertson1968(CCT_D_uv)
array([ 0.1937413...,  0.3152210...])
```

Krystek (1985)

colour.temperature

uv_to_CCT_Krystek1985(uv[, ...])	Returns the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Krystek (1985)* method.
CCT_to_uv_Krystek1985(CCT)	Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

colour.temperature.uv_to_CCT_Krystek1985

colour.temperature.**uv_to_CCT_Krystek1985**(*uv*, *optimisation_parameters=None*)

Returns the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates using *Krystek (1985)* method.

Parameters

- **uv** (array_like) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type ndarray

Warning: *Krystek (1985)* does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE UCS* colourspace *uv* chromaticity coordinates, the current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} normalised to domain [1000, 15000].

References

[Kry85]

Examples

```
>>> uv_to_CCT_Krystek1985(np.array([0.20047203, 0.31029290]))
...
6504.3894290...
```

colour.temperature.CCT_to_uv_Krystek1985

colour.temperature.**CCT_to_uv_Krystek1985**(*CCT*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp} using *Krystek (1985)* method.

Parameters **CCT** (array_like) – Correlated colour temperature T_{cp}.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

Notes

- *Krystek (1985)* method computations are valid for correlated colour temperature T_{cp} normalised to domain [1000, 15000].

References

[Kry85]

Examples

```
>>> CCT_to_uv_Krystek1985(6504.38938305)
array([ 0.2004720...,  0.3102929...])
```

Ohno (2013)

colour.temperature

uv_to_CCT_Ohno2013(uv[, cmfs, start, end, ...])	Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates, colour matching functions and temperature range using *Ohno (2013)* method.

Continued on next page

Table 238 – continued from previous page

CCT_to_uv_Ohno2013(CCT_D_uv[, cmfs])	Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp}, Δ_{uv} and colour matching functions using *Ohno (2013)* method.

colour.temperature.uv_to_CCT_Ohno2013

colour.temperature.**uv_to_CCT_Ohno2013**(*uv*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *start=1000*, *end=100000*, *count=10*, *iterations=6*)

Returns the correlated colour temperature T_{cp} and Δ_{uv} from given *CIE UCS* colourspace *uv* chromaticity coordinates, colour matching functions and temperature range using *Ohno (2013)* method.

The iterations parameter defines the calculations precision: The higher its value, the more planckian tables will be generated through cascade expansion in order to converge to the exact solution.

Parameters

- **uv** (array_like) – *CIE UCS* colourspace *uv* chromaticity coordinates.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **start** (numeric, optional) – Temperature range start in kelvins.
- **end** (numeric, optional) – Temperature range end in kelvins.
- **count** (int, optional) – Temperatures count in the planckian tables.
- **iterations** (int, optional) – Number of planckian tables to generate.

Returns Correlated colour temperature T_{cp}, Δ_{uv}.

Return type ndarray

References

[Ohn14]

Examples

```
>>> from colour import DEFAULT_SPECTRAL_SHAPE, STANDARD_OBSERVERS_CMFS
>>> cmfs = (
...     STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer'].
...     copy().align(DEFAULT_SPECTRAL_SHAPE)
... )
>>> uv = np.array([0.1978, 0.3122])
>>> uv_to_CCT_Ohno2013(uv, cmfs)
array([  6.5074738...e+03,   3.2233461...e-03])
```

colour.temperature.CCT_to_uv_Ohno2013

colour.temperature.**CCT_to_uv_Ohno2013**(*CCT_D_uv*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*)

Returns the *CIE UCS* colourspace *uv* chromaticity coordinates from given correlated colour temperature T_{cp}, Δ_{uv} and colour matching functions using *Ohno (2013)* method.

Parameters

- **CCT_D_uv** (ndarray) – Correlated colour temperature T_{cp}, Δ_{uv}.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.

Returns *CIE UCS* colourspace *uv* chromaticity coordinates.

Return type ndarray

References

[Ohn14]

Examples

```
>>> from colour import DEFAULT_SPECTRAL_SHAPE, STANDARD_OBSERVERS_CMFS
>>> cmfs = (
...     STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer'].
...     copy().align(DEFAULT_SPECTRAL_SHAPE)
... )
>>> CCT_D_uv = np.array([6507.4342201047066, 0.003223690901513])
>>> CCT_to_uv_Ohno2013(CCT_D_uv, cmfs)
array([ 0.1977999...,  0.3122004...])
```

McCamy (1992)

colour.temperature

xy_to_CCT_McCamy1992(xy)	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *McCamy (1992)* method.
CCT_to_xy_McCamy1992(CCT[, ...])	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *McCamy (1992)* method.

colour.temperature.xy_to_CCT_McCamy1992

colour.temperature.**xy_to_CCT_McCamy1992**(*xy*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *McCamy (1992)* method.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type numeric or ndarray

References

[Wik01a]

Examples

```
>>> import numpy as np
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_McCamy1992(xy)
6505.0805913...
```

colour.temperature.CCT_to_xy_McCamy1992

colour.temperature.**CCT_to_xy_McCamy1992**(*CCT*, *optimisation_parameters=None*)

Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *McCamy (1992)* method.

Parameters

- **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Warning: *McCamy (1992)* method for computing *CIE xy* chromaticity coordinates from given correlated colour temperature is a bijective function and might produce unexpected results. It is given for consistency with other correlated colour temperature computation methods but should be avoided for practical applications.The current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[Wik01a]

Examples

```
>>> CCT_to_xy_McCamy1992(6505.0805913074782)
array([ 0.3127...,  0.329...])
```

Hernandez-Andres, Lee and Romero (1999)

colour.temperature

xy_to_CCT_Hernandez1999(xy)	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Hernandez-Andres et al.(1999)* method..
CCT_to_xy_Hernandez1999(CCT[, ...])	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Hernandez-Andres et al.(1999)* method..

colour.temperature.xy_to_CCT_Hernandez1999

colour.temperature.**xy_to_CCT_Hernandez1999**(*xy*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Hernandez-Andres et al. (1999)* method.

Parameters **xy** (array_like) – *CIE xy* chromaticity coordinates.

Returns Correlated colour temperature T_{cp}.

Return type numeric

References

[HernandezAndresLR99]

Examples

```
>>> xy = np.array([0.31270, 0.32900])
>>> xy_to_CCT_Hernandez1999(xy)
6500.7420431...
```

colour.temperature.CCT_to_xy_Hernandez1999

colour.temperature.**CCT_to_xy_Hernandez1999**(*CCT*, *optimisation_parameters=None*)

Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Hernandez-Andres et al. (1999)* method.

Parameters

- **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Warning: *Hernandez-Andres et al. (1999)* method for computing *CIE xy* chromaticity coordinates from given correlated colour temperature is not a bijective function and might produce unexpected results. It is given for consistency with other correlated colour temperature computation methods but should be avoided for practical applications. The current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[HernandezAndresLR99]

Examples

```
>>> CCT_to_xy_Hernandez1999(6500.7420431786531)
array([ 0.3127...,  0.329...])
```

Kang, Moon, Hong, Lee, Cho and Kim (2002)

colour.temperature

xy_to_CCT_Kang2002(xy[, optimisation_parameters])	Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Kang et al.(2002)* method..
CCT_to_xy_Kang2002(CCT)	Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et al.(2002)* method..

colour.temperature.xy_to_CCT_Kang2002

colour.temperature.**xy_to_CCT_Kang2002**(*xy*, *optimisation_parameters=None*)

Returns the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates using *Kang et al. (2002)* method.

Parameters

- **xy** (array_like) – *CIE xy* chromaticity coordinates.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type ndarray

Warning: *Kang et al. (2002)* does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates, the current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[KMH+02]

Examples

```
>>> xy_to_CCT_Kang2002(np.array([0.31342600, 0.32359597]))
...
6504.3893128...
```

colour.temperature.CCT_to_xy_Kang2002

colour.temperature.**CCT_to_xy_Kang2002**(*CCT*)

Returns the *CIE xy* chromaticity coordinates from given correlated colour temperature T_{cp} using *Kang et al. (2002)* method.

Parameters **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Raises ValueError – If the correlated colour temperature is not in appropriate domain.

References

[KMH+02]

Examples

```
>>> CCT_to_xy_Kang2002(6504.38938305)
array([ 0.313426 ...,  0.3235959...])
```

CIE Illuminant D Series

colour.temperature

xy_to_CCT_CIE_D(xy[, optimisation_parameters])	Returns the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* from its *CIE xy* chromaticity coordinates.
CCT_to_xy_CIE_D(CCT)	Returns the *CIE xy* chromaticity coordinates of a *CIE Illuminant D Series* from its correlated colour temperature T_{cp}.

colour.temperature.xy_to_CCT_CIE_D

colour.temperature.**xy_to_CCT_CIE_D**(*xy*, *optimisation_parameters=None*)

Returns the correlated colour temperature T_{cp} of a *CIE Illuminant D Series* from its *CIE xy* chromaticity coordinates.

Parameters

- **xy** (array_like) – *CIE xy* chromaticity coordinates.
- **optimisation_parameters** (dict_like, optional) – Parameters for scipy.optimize.minimize() definition.

Returns Correlated colour temperature T_{cp}.

Return type ndarray

Warning: The *CIE Illuminant D Series* method does not give an analytical inverse transformation to compute the correlated colour temperature T_{cp} from given *CIE xy* chromaticity coordinates, the current implementation relies on optimization using scipy.optimize.minimize() definition and thus has reduced precision and poor performance.

References

[WS00d]

Examples

```
>>> xy_to_CCT_CIE_D(np.array([0.31270775, 0.32911283]))
...
6504.3895840...
```

colour.temperature.CCT_to_xy_CIE_D

colour.temperature.**CCT_to_xy_CIE_D**(*CCT*)

Returns the *CIE xy* chromaticity coordinates of a *CIE Illuminant D Series* from its correlated colour temperature T_{cp}.

Parameters **CCT** (numeric or array_like) – Correlated colour temperature T_{cp}.

Returns *CIE xy* chromaticity coordinates.

Return type ndarray

Raises ValueError – If the correlated colour temperature is not in appropriate domain.

References

[WS00d]

Examples

```
>>> CCT_to_xy_CIE_D(6504.38938305)
array([ 0.3127077...,  0.3291128...])
```

Utilities

- *Common*
- *Array*
- *Metrics*
- *Data Structures*
- *Verbose*

Common

colour

domain_range_scale(scale)	A context manager and decorator temporarily setting *Colour* domain-range scale.
get_domain_range_scale()	Returns the current *Colour* domain-range scale.
set_domain_range_scale([scale])	Sets the current *Colour* domain-range scale.

colour.domain_range_scale

class colour.**domain_range_scale**(*scale*)

A context manager and decorator temporarily setting *Colour* domain-range scale. The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc...
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Parameters **scale** (unicode) – **{'Reference', '1'}**, *Colour* domain-range scale to set.

__init__(*scale*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(scale)	Initialize self.

colour.get_domain_range_scale

colour.**get_domain_range_scale**()

Returns the current *Colour* domain-range scale. The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc...
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Returns *Colour* domain-range scale.

Return type unicode

colour.set_domain_range_scale

colour.**set_domain_range_scale**(*scale='Reference'*)

Sets the current *Colour* domain-range scale. The following scales are available:

- **'Reference'**, the default *Colour* domain-range scale which varies depending on the referenced algorithm, e.g. [0, 1], [0, 10], [0, 100], [0, 255], etc...
- **'1'**, a domain-range scale normalised to [0, 1], it is important to acknowledge that this is a soft normalisation and it is possible to use negative out of gamut values or high dynamic range data exceeding 1.

Parameters **scale** (unicode or int) – **{'Reference', '1'}**, *Colour* domain-range scale to set.

colour.utilities

handle_numpy_errors(**kwargs)	Decorator for handling *Numpy* errors.
ignore_numpy_errors(function)	Wrapper for given function.
raise_numpy_errors(function)	Wrapper for given function.
print_numpy_errors(function)	Wrapper for given function.
warn_numpy_errors(function)	Wrapper for given function.
ignore_python_warnings(function)	Decorator for ignoring *Python* warnings.
batch(iterable[, k])	Returns a batch generator from given iterable.
disable_multiprocessing	A context manager and decorator temporarily disabling *Colour* multiprocessing.
multiprocessing_pool(*args, **kwargs)	A context manager providing a multiprocessing pool.
is_matplotlib_installed([raise_exception])	Returns if *Matplotlib* is installed and available.
is_networkx_installed([raise_exception])	Returns if *NetworkX* is installed and available.
is_openimageio_installed([raise_exception])	Returns if *OpenImageIO* is installed and available.
is_pandas_installed([raise_exception])	Returns if *Pandas* is installed and available.
is_iterable(a)	Returns if given a variable is iterable.
is_string(a)	Returns if given a variable is a *string* like variable.
is_numeric(a)	Returns if given a variable is a number.
is_integer(a)	Returns if given a variable is an integer under given threshold.
is_sibling(element, mapping)	Returns whether given element type is present in given mapping types.
filter_kwargs(function, **kwargs)	Filters keyword arguments incompatible with the given function signature.
filter_mapping(mapping, filterers[, ...])	Filters given mapping with given filterers.
first_item(a)	Return the first item of an iterable.
to_domain_1(a[, scale_factor, dtype])	Scales given array a to domain **'1'**.
to_domain_10(a[, scale_factor, dtype])	Scales given array a to domain **'10'**, used by *Munsell Renotation System*.
to_domain_100(a[, scale_factor, dtype])	Scales given array a to domain **'100'**.
to_domain_degrees(a[, scale_factor, dtype])	Scales given array a to degrees domain.
to_domain_int(a[, bit_depth, dtype])	Scales given array a to int domain.
from_range_1(a[, scale_factor])	Scales given array a from range **'1'**.
from_range_10(a[, scale_factor])	Scales given array a from range **'10'**, used by *Munsell Renotation System*.
from_range_100(a[, scale_factor])	Scales given array a from range **'100'**.
from_range_degrees(a[, scale_factor])	Scales given array a from degrees range.
from_range_int(a[, bit_depth, dtype])	Scales given array a from int range.

colour.utilities.handle_numpy_errors

colour.utilities.**handle_numpy_errors**(***kwargs*)
Decorator for handling *Numpy* errors.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns

Return type object

References

[KPK11]

Examples

```
>>> import numpy
>>> @handle_numpy_errors(all='ignore')
... def f():
...     1 / numpy.zeros(3)
>>> f()
```

colour.utilities.ignore_numpy_errors

colour.utilities.**ignore_numpy_errors**(*function*)
Wrapper for given function.

colour.utilities.raise_numpy_errors

colour.utilities.**raise_numpy_errors**(*function*)
Wrapper for given function.

colour.utilities.print_numpy_errors

colour.utilities.**print_numpy_errors**(*function*)
Wrapper for given function.

colour.utilities.warn_numpy_errors

colour.utilities.**warn_numpy_errors**(*function*)
Wrapper for given function.

colour.utilities.ignore_python_warnings

colour.utilities.**ignore_python_warnings**(*function*)

Decorator for ignoring *Python* warnings.

Parameters **function** (object) – Function to decorate.

Returns

Return type object

Examples

```
>>> @ignore_python_warnings
... def f():
...     warnings.warn('This is an ignored warning!')
>>> f()
```

colour.utilities.batch

colour.utilities.**batch**(*iterable*, *k=3*)

Returns a batch generator from given iterable.

Parameters

- **iterable** (iterable) – Iterable to create batches from.
- **k** (integer) – Batches size.

Returns Is *string_like* variable.

Return type bool

Examples

```
>>> batch(tuple(range(10)))
<generator object batch at 0x...>
```

colour.utilities.disable_multiprocessing

class colour.utilities.**disable_multiprocessing**

A context manager and decorator temporarily disabling *Colour* multiprocessing.

__init__()

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__	Initialize self.

colour.utilities.multiprocessing_pool

colour.utilities.**multiprocessing_pool**(**args*, ***kwargs*)
A context manager providing a multiprocessing pool.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Keywords arguments.

Examples

```
>>> from functools import partial
>>> def _add(a, b):
...     return a + b
>>> with multiprocessing_pool() as pool:
...     pool.map(partial(_add, b=2), range(10))
...
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

colour.utilities.is_matplotlib_installed

colour.utilities.**is_matplotlib_installed**(*raise_exception=False*)
Returns if *Matplotlib* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *Matplotlib* is unavailable.

Returns Is *Matplotlib* installed.

Return type bool

Raises **ImportError** – If *Matplotlib* is not installed.

colour.utilities.is_networkx_installed

colour.utilities.**is_networkx_installed**(*raise_exception=False*)
Returns if *NetworkX* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *NetworkX* is unavailable.

Returns Is *NetworkX* installed.

Return type bool

Raises **ImportError** – If *NetworkX* is not installed.

colour.utilities.is_openimageio_installed

colour.utilities.**is_openimageio_installed**(*raise_exception=False*)
Returns if *OpenImageIO* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *OpenImageIO* is unavailable.

Returns Is *OpenImageIO* installed.

Return type bool

Raises **ImportError** – If *OpenImageIO* is not installed.

colour.utilities.is_pandas_installed

colour.utilities.**is_pandas_installed**(*raise_exception=False*)

Returns if *Pandas* is installed and available.

Parameters **raise_exception** (bool) – Raise exception if *Pandas* is unavailable.

Returns Is *Pandas* installed.

Return type bool

Raises **ImportError** – If *Pandas* is not installed.

colour.utilities.is_iterable

colour.utilities.**is_iterable**(*a*)

Returns if given a variable is iterable.

Parameters **a** (object) – Variable to check the iterability.

Returns a variable iterability.

Return type bool

Examples

```
>>> is_iterable([1, 2, 3])
True
>>> is_iterable(1)
False
```

colour.utilities.is_string

colour.utilities.**is_string**(*a*)

Returns if given a variable is a *string* like variable.

Parameters **a** (object) – Data to test.

Returns Is a variable a *string* like variable.

Return type bool

Examples

```
>>> is_string("I'm a string!")
True
>>> is_string(["I'm a string!"])
False
```

colour.utilities.is_numeric

colour.utilities.**is_numeric**(*a*)

Returns if given a variable is a number.

Parameters **a** (object) – Variable to check.

Returns Is a variable a number.

Return type bool

Examples

```
>>> is_numeric(1)
True
>>> is_numeric((1,))
False
```

colour.utilities.is_integer

colour.utilities.**is_integer**(*a*)

Returns if given a variable is an integer under given threshold.

Parameters **a** (object) – Variable to check.

Returns Is a variable an integer.

Return type bool

Notes

- The determination threshold is defined by the colour.algebra.common.INTEGER_THRESHOLD attribute.

Examples

```
>>> is_integer(1)
True
>>> is_integer(1.01)
False
```

colour.utilities.is_sibling

colour.utilities.**is_sibling**(*element*, *mapping*)

Returns whether given element type is present in given mapping types.

Parameters

- **element** (object) – Element to check if its type is present in the mapping types.
- **mapping** (dict) – Mapping.

Returns Whether given element type is present in given mapping types.

Return type bool

colour.utilities.filter_kwargs

colour.utilities.**filter_kwargs**(*function*, ***kwargs*)

Filters keyword arguments incompatible with the given function signature.

Parameters **function** (callable) – Callable to filter the incompatible keyword arguments.

Other Parameters ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Filtered keyword arguments.

Return type dict

Warning: Python 2.7 does not support inspecting the signature of *partial* functions, this could cause unexpected behaviour when using this definition.

Examples

```
>>> def fn_a(a):
...     return a
>>> def fn_b(a, b=0):
...     return a, b
>>> def fn_c(a, b=0, c=0):
...     return a, b, c
>>> fn_a(1, **filter_kwargs(fn_a, b=2, c=3))
1
>>> fn_b(1, **filter_kwargs(fn_b, b=2, c=3))
(1, 2)
>>> fn_c(1, **filter_kwargs(fn_c, b=2, c=3))
(1, 2, 3)
```

colour.utilities.filter_mapping

colour.utilities.**filter_mapping**(*mapping*, *filterers*, *anchors=True*, *flags=<RegexFlag.IGNORECASE: 2>*)

Filters given mapping with given filterers.

Parameters

- **mapping** (dict_like) – Mapping to filter.
- **filterers** (unicode or object or array_like) – Filterer pattern for given mapping elements or a list of filterers.
- **anchors** (bool, optional) – Whether to use Regex line anchors, i.e. *^* and *$* are added, surrounding the filterer pattern.
- **flags** (int, optional) – Regex flags.

Returns Filtered mapping elements.

Return type OrderedDict

Notes

- To honour the filterers ordering, the return value is an `OrderedDict` class instance.

Examples

```
>>> class Element(object):
...     pass
>>> mapping = {
...     'Element A': Element(),
...     'Element B': Element(),
...     'Element C': Element(),
...     'Not Element C': Element(),
... }
>>> # Doctests skip for Python 2.x compatibility.
>>> filter_mapping(mapping, '\w+\s+A')
{u'Element A': <colour.utilities.common.Element object at 0x...>}
>>> # Doctests skip for Python 2.x compatibility.
>>> sorted(filter_mapping(mapping, 'Element.*'))
[u'Element A', u'Element B', u'Element C']
```

colour.utilities.first_item

`colour.utilities.`**`first_item`**(*a*)

Return the first item of an iterable.

Parameters **a** (`object`) – Iterable to get the first item from.

Returns

Return type object

Raises `StopIteration` – If the iterable is empty.

Examples

```
>>> a = range(10)
>>> first_item(a)
0
```

colour.utilities.to_domain_1

`colour.utilities.`**`to_domain_1`**(*a*, *scale_factor=100*, *dtype=<class 'numpy.float64'>*)

Scales given array a to domain **'1'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'1'**, the definition is almost entirely by-passed and will just conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is divided by `scale_factor`, typically 100.

Parameters

- **a** (`array_like`) – a to scale to domain **'1'**.
- **scale_factor** (`numeric or array_like, optional`) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to domain **'1'**.

- **dtype** (object, optional) – Data type used for the conversion to np.ndarray.

Returns a scaled to domain **'1'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     to_domain_1(1)
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     to_domain_1(1)
array(1.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_1(1)
array(0.01)
```

colour.utilities.to_domain_10

colour.utilities.**to_domain_10**(*a*, *scale_factor=10*, *dtype=<class 'numpy.float64'>*)

Scales given array a to domain **'10'**, used by *Munsell Renotation System*. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is almost entirely by-passed and will just conveniently convert array a to np.ndarray.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by scale_factor, typically 10.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is divided by scale_factor, typically 10.

Parameters

- **a** (array_like) – a to scale to domain **'10'**.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to domain **'10'**.
- **dtype** (object, optional) – Data type used for the conversion to np.ndarray.

Returns a scaled to domain **'10'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     to_domain_10(1)
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     to_domain_10(1)
array(10.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_10(1)
array(0.1)
```

colour.utilities.to_domain_100

colour.utilities.**to_domain_100**(*a*, *scale_factor=100*, *dtype=<class 'numpy.float64'>*)

Scales given array a to domain **'100'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'100'** (currently unsupported private value only used for unit tests), the definition is almost entirely by-passed and will just conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by `scale_factor`, typically 100.

Parameters

- **a** (`array_like`) – a to scale to domain **'100'**.
- **scale_factor** (`numeric or array_like, optional`) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to domain **'100'**.
- **dtype** (`object, optional`) – Data type used for the conversion to `np.ndarray`.

Returns a scaled to domain **'100'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     to_domain_100(1)
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     to_domain_100(1)
array(100.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_100(1)
array(1.0)
```

colour.utilities.to_domain_degrees

colour.utilities.**to_domain_degrees**(*a*, *scale_factor=360*, *dtype=<class 'numpy.float64'>*)

Scales given array a to degrees domain. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is almost entirely by-passed and will just conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by `scale_factor`, typically 360.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by `scale_factor / 100`, typically 360 / 100.

Parameters

- **a** (`array_like`) – a to scale to degrees domain.
- **scale_factor** (`numeric or array_like, optional`) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought to degrees domain.
- **dtype** (`object, optional`) – Data type used for the conversion to `np.ndarray`.

Returns a scaled to degrees domain.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     to_domain_degrees(1)
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     to_domain_degrees(1)
array(360.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_degrees(1)
array(3.6)
```

colour.utilities.to_domain_int

colour.utilities.**to_domain_int**(*a*, *bit_depth=8*, *dtype=<class 'numpy.float64'>*)

Scales given array a to int domain. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is almost entirely by-passed and will just conveniently convert array a to `np.ndarray`.
- If *Colour* domain-range scale is **'1'**, array a is multiplied by $2^{bit\_depth} - 1$.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by $2^{bit\_depth} - 1$.

Parameters

- **a** (`array_like`) – a to scale to int domain.
- **bit_depth** (`numeric or array_like, optional`) – Bit depth, usually *int* but can be an *array_like* if some axis need different scaling to be brought to int domain.
- **dtype** (`object, optional`) – Data type used for the conversion to `np.ndarray`.

Returns a scaled to int domain.

Return type ndarray

Notes

- To avoid precision issues and rounding, the scaling is performed on floating-point numbers.

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     to_domain_int(1)
array(1.0)
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     to_domain_int(1)
array(255.0)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     to_domain_int(1)
array(2.55)
```

colour.utilities.from_range_1

colour.utilities.**from_range_1**(*a*, *scale_factor=100*)

Scales given array a from range **'1'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'1'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by `scale_factor`, typically 100.

Parameters

- **a** (array_like) – a to scale from range **'1'**.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from range **'1'**.

Returns a scaled from range **'1'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     from_range_1(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     from_range_1(1)
1
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_1(1)
100
```

colour.utilities.from_range_10

colour.utilities.**from_range_10**(*a*, *scale_factor=10*)

Scales given array a from range **'10'**, used by *Munsell Renotation System*. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is divided by scale_factor, typically 10.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is multiplied by scale_factor, typically 10.

Parameters

- **a** (array_like) – a to scale from range **'10'**.
- **scale_factor** (numeric or array_like, optional) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from range **'10'**.

Returns a scaled from range **'10'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     from_range_10(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     from_range_10(1)
0.1
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_10(1)
10
```

colour.utilities.from_range_100

colour.utilities.**from_range_100**(*a*, *scale_factor=100*)

Scales given array a from range **'100'**. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'** or **'100'** (currently unsupported private value only used for unit tests), the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is divided by `scale_factor`, typically 100.

Parameters

- **a** (`array_like`) – a to scale from range **'100'**.
- **scale_factor** (`numeric or array_like, optional`) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from range **'100'**.

Returns a scaled from range **'100'**.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     from_range_100(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     from_range_100(1)
0.01
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_100(1)
1
```

colour.utilities.from_range_degrees

colour.utilities.**from_range_degrees**(*a*, *scale_factor=360*)

Scales given array a from degrees range. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is divided by `scale_factor`, typically 360.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is divided by `scale_factor` / 100, typically 360 / 100.

Parameters

- **a** (`array_like`) – a to scale from degrees range.
- **scale_factor** (`numeric or array_like, optional`) – Scale factor, usually *numeric* but can be an *array_like* if some axis need different scaling to be brought from degrees range.

Returns a scaled from degrees range.

Return type ndarray

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     from_range_degrees(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     from_range_degrees(1)
0.0027777...
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_degrees(1)
0.2777777...
```

colour.utilities.from_range_int

colour.utilities.**from_range_int**(*a*, *bit_depth=8*, *dtype=<class 'numpy.float64'>*)

Scales given array a from int range. The behaviour is as follows:

- If *Colour* domain-range scale is **'Reference'**, the definition is entirely by-passed.
- If *Colour* domain-range scale is **'1'**, array a is converted to `np.ndarray` and divided by $2^{bit\_depth} - 1$.
- If *Colour* domain-range scale is **'100'** (currently unsupported private value only used for unit tests), array a is converted to `np.ndarray` and divided by $2^{bit\_depth} - 1$.

Parameters

- **a** (`array_like`) – a to scale from int range.
- **bit_depth** (`numeric or array_like, optional`) – Bit depth, usually *int* but can be an *array_like* if some axis need different scaling to be brought from int range.

- **dtype** (object, optional) – Data type used for the conversion to np.ndarray.

Returns a scaled from int range.

Return type ndarray

Notes

- To avoid precision issues and rounding, the scaling is performed on floating-point numbers.

Examples

With *Colour* domain-range scale set to **'Reference'**:

```
>>> with domain_range_scale('Reference'):
...     from_range_int(1)
1
```

With *Colour* domain-range scale set to **'1'**:

```
>>> with domain_range_scale('1'):
...     from_range_int(1)
array(0.0039215...)
```

With *Colour* domain-range scale set to **'100'** (unsupported):

```
>>> with domain_range_scale('100'):
...     from_range_int(1)
array(0.3921568...)
```

Array

colour.utilities

as_array(a[, dtype])	Converts given a variable to *ndarray* with given type.
as_int_array(a)	Converts given a variable to *ndarray* using the type defined by colour.constant.DEFAULT_INT_DTYPE attribute.
as_float_array(a)	Converts given a variable to *ndarray* using the type defined by colour.constant.DEFAULT_FLOAT_DTYPE attribute.
as_numeric(a[, dtype])	Converts given a variable to *numeric*.
as_int(a)	Converts given a variable to *numeric* using the type defined by colour.constant.DEFAULT_INT_DTYPE attribute.
as_float(a)	Converts given a variable to *numeric* using the type defined by colour.constant.DEFAULT_FLOAT_DTYPE attribute.
as_namedtuple(a, named_tuple)	Converts given a variable to given *namedtuple* class instance.
closest_indexes(a, b)	Returns the a variable closest element indexes to reference b variable elements.
closest(a, b)	Returns the a variable closest elements to reference b variable elements.

Continued on next page

Table 247 – continued from previous page

normalise_maximum(a[, axis, factor, clip])	Normalises given *array_like* a variable values by a variable maximum value and optionally clip them between.
interval(distribution[, unique])	Returns the interval size of given distribution.
is_uniform(distribution)	Returns if given distribution is uniform.
in_array(a, b[, tolerance])	Tests whether each element of an array is also present in a second array within given tolerance.
tstack(a[, dtype])	Stacks arrays in sequence along the last axis (tail).
tsplit(a[, dtype])	Splits arrays in sequence along the last axis (tail).
row_as_diagonal(a)	Returns the per row diagonal matrices of the given array.
dot_vector(m, v)	Convenient wrapper around np.einsum() with the following subscripts: *'...ij,...j->...i'*.
dot_matrix(a, b)	Convenient wrapper around np.einsum() with the following subscripts: *'...ij,...jk->...ik'*.
orient(a, orientation)	Orient given array according to given orientation value.
centroid(a)	Computes the centroid indexes of given a array.
linear_conversion(a, old_range, new_range)	Performs a simple linear conversion of given array between the old and new ranges.
lerp(a, b, c)	Performs a simple linear interpolation between given array a and array b using c value.
fill_nan(a[, method, default])	Fills given array NaNs according to given method.
ndarray_write(a)	A context manager setting given array writeable to perform an operation and then read-only.

colour.utilities.as_array

colour.utilities.**as_array**(*a*, *dtype=<class 'numpy.float64'>*)

Converts given a variable to *ndarray* with given type.

Parameters

- **a** (object) – Variable to convert.
- **dtype** (object) – Type to use for conversion.

Returns a variable converted to *ndarray*.

Return type ndarray

Examples

```
>>> as_array([1, 2, 3])
array([ 1.,  2.,  3.])
>>> as_array([1, 2, 3], dtype=DEFAULT_INT_DTYPE)
array([1, 2, 3])
```

colour.utilities.as_int_array

colour.utilities.**as_int_array**(*a*)

Converts given a variable to *ndarray* using the type defined by `colour.constant.DEFAULT_INT_DTYPE` attribute.

Parameters **a** (`object`) – Variable to convert.

Returns a variable converted to *ndarray*.

Return type ndarray

Examples

```
>>> as_int_array([1.0, 2.0, 3.0])
array([1, 2, 3])
```

colour.utilities.as_float_array

colour.utilities.**as_float_array**(*a*)

Converts given a variable to *ndarray* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Parameters **a** (`object`) – Variable to convert.

Returns a variable converted to *ndarray*.

Return type ndarray

Examples

```
>>> as_float_array([1, 2, 3])
array([ 1.,  2.,  3.])
```

colour.utilities.as_numeric

colour.utilities.**as_numeric**(*a*, *dtype=<class 'numpy.float64'>*)

Converts given a variable to *numeric*. In the event where a cannot be converted, it is passed as is.

Parameters

- **a** (`object`) – Variable to convert.
- **dtype** (`object`) – Type to use for conversion.

Returns a variable converted to *numeric*.

Return type ndarray

Examples

```
>>> as_numeric(np.array([1]))
1.0
>>> as_numeric(np.arange(10))
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

colour.utilities.as_int

colour.utilities.**as_int**(*a*)

Converts given a variable to *numeric* using the type defined by `colour.constant.DEFAULT_INT_DTYPE` attribute. In the event where a cannot be converted, it is converted to *ndarray* using the type defined by `colour.constant.DEFAULT_INT_DTYPE` attribute.

Parameters **a** (`object`) – Variable to convert.

Returns a variable converted to *numeric*.

Return type ndarray

Warning: The behaviour of this definition is different than `colour.utilities.as_numeric()` definition when it comes to conversion failure: the former will forcibly convert a variable to *ndarray* using the type defined by `colour.constant.DEFAULT_INT_DTYPE` attribute while the later will pass the a variable as is.

Examples

```
>>> as_int(np.array([1]))
1
>>> as_int(np.arange(10))
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

colour.utilities.as_float

colour.utilities.**as_float**(*a*)

Converts given a variable to *numeric* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute. In the event where a cannot be converted, it is converted to *ndarray* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute.

Parameters **a** (`object`) – Variable to convert.

Returns a variable converted to *numeric*.

Return type ndarray

Warning: The behaviour of this definition is different than `colour.utilities.as_numeric()` definition when it comes to conversion failure: the former will forcibly convert a variable to *ndarray* using the type defined by `colour.constant.DEFAULT_FLOAT_DTYPE` attribute while the later will pass the a variable as is.

Examples

```
>>> as_float(np.array([1]))
1.0
>>> as_float(np.arange(10))
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

colour.utilities.as_namedtuple

colour.utilities.**as_namedtuple**(*a*, *named_tuple*)

Converts given a variable to given *namedtuple* class instance.

a can be either a *Numpy* structured array, a *namedtuple*, a *mapping*, or an *array_like* object. The definition will attempt to convert it to given *namedtuple*.

Parameters

- **a** (object) – Variable to convert.
- **named_tuple** (namedtuple) – *namedtuple* class.

Returns math:*a* variable converted to *namedtuple*.

Return type namedtuple

Examples

```
>>> from collections import namedtuple
>>> a_a = 1
>>> a_b = 2
>>> a_c = 3
>>> NamedTuple = namedtuple('NamedTuple', 'a b c')
>>> as_namedtuple(NamedTuple(a=1, b=2, c=3), NamedTuple)
NamedTuple(a=1, b=2, c=3)
>>> as_namedtuple({'a': a_a, 'b': a_b, 'c': a_c}, NamedTuple)
NamedTuple(a=1, b=2, c=3)
>>> as_namedtuple([a_a, a_b, a_c], NamedTuple)
NamedTuple(a=1, b=2, c=3)
```

colour.utilities.closest_indexes

colour.utilities.**closest_indexes**(*a*, *b*)

Returns the a variable closest element indexes to reference b variable elements.

Parameters

- **a** (array_like) – Variable to search for the closest element indexes.
- **b** (numeric) – Reference variable.

Returns Closest a variable element indexes.

Return type numeric

Examples

```
>>> a = np.array([24.31357115, 63.62396289, 55.71528816,
...               62.70988028, 46.84480573, 25.40026416])
>>> print(closest_indexes(a, 63))
[3]
>>> print(closest_indexes(a, [63, 25]))
[3 5]
```

colour.utilities.closest

colour.utilities.**closest**(*a*, *b*)

Returns the a variable closest elements to reference b variable elements.

Parameters

- **a** (array_like) – Variable to search for the closest elements.
- **b** (numeric) – Reference variable.

Returns Closest a variable elements.

Return type numeric

Examples

```
>>> a = np.array([24.31357115, 63.62396289, 55.71528816,
...               62.70988028, 46.84480573, 25.40026416])
>>> closest(a, 63)
array([ 62.70988028])
>>> closest(a, [63, 25])
array([ 62.70988028,  25.40026416])
```

colour.utilities.normalise_maximum

colour.utilities.**normalise_maximum**(*a*, *axis=None*, *factor=1*, *clip=True*)

Normalises given *array_like* a variable values by a variable maximum value and optionally clip them between.

Parameters

- **a** (array_like) – a variable to normalise.
- **axis** (numeric, optional) – Normalization axis.
- **factor** (numeric, optional) – Normalization factor.
- **clip** (bool, optional) – Clip values to domain [0, 'factor'].

Returns Maximum normalised a variable.

Return type ndarray

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> normalise_maximum(a)
array([ 1.        ,  0.6564384...,  0.4576822...])
```

colour.utilities.interval

colour.utilities.**interval**(*distribution*, *unique=True*)

Returns the interval size of given distribution.

Parameters

- **distribution** (array_like) – Distribution to retrieve the interval.
- **unique** (bool, optional) – Whether to return unique intervals if the distribution is non-uniformly spaced or the complete intervals

Returns Distribution interval.

Return type ndarray

Examples

Uniformly spaced variable:

```
>>> y = np.array([1, 2, 3, 4, 5])
>>> interval(y)
array([ 1.])
>>> interval(y, False)
array([ 1.,  1.,  1.,  1.])
```

Non-uniformly spaced variable:

```
>>> y = np.array([1, 2, 3, 4, 8])
>>> interval(y)
array([ 1.,  4.])
>>> interval(y, False)
array([ 1.,  1.,  1.,  4.])
```

colour.utilities.is_uniform

colour.utilities.**is_uniform**(*distribution*)

Returns if given distribution is uniform.

Parameters **distribution** (array_like) – Distribution to check for uniformity.

Returns Is distribution uniform.

Return type bool

Examples

Uniformly spaced variable:

```
>>> a = np.array([1, 2, 3, 4, 5])
>>> is_uniform(a)
True
```

Non-uniformly spaced variable:

```
>>> a = np.array([1, 2, 3.1415, 4, 5])
>>> is_uniform(a)
False
```

colour.utilities.in_array

colour.utilities.**in_array**(*a*, *b*, *tolerance=2.2204460492503131e-16*)

Tests whether each element of an array is also present in a second array within given tolerance.

Parameters

- **a** (array_like) – Array to test the elements from.
- **b** (array_like) – The values against which to test each value of array *a*.
- **tolerance** (numeric, optional) – Tolerance value.

Returns A boolean array with *a* shape describing whether an element of *a* is present in *b* within given tolerance.

Return type ndarray

References

[Yor14]

Examples

```
>>> a = np.array([0.50, 0.60])
>>> b = np.linspace(0, 10, 101)
>>> np.in1d(a, b)
array([ True, False], dtype=bool)
>>> in_array(a, b)
array([ True,  True], dtype=bool)
```

colour.utilities.tstack

colour.utilities.**tstack**(*a*, *dtype=<class 'numpy.float64'>*)

Stacks arrays in sequence along the last axis (tail).

Rebuilds arrays divided by colour.utilities.tsplit().

Parameters

- **a** (array_like) – Array to perform the stacking.
- **dtype** (object) – Type to use for initial conversion to *ndarray*.

Returns

Return type ndarray

Examples

```
>>> a = 0
>>> tstack([a, a, a])
array([ 0.,  0.,  0.])
>>> a = np.arange(0, 6)
>>> tstack([a, a, a])
array([[ 0.,  0.,  0.],
       [ 1.,  1.,  1.],
       [ 2.,  2.,  2.],
       [ 3.,  3.,  3.],
       [ 4.,  4.,  4.],
       [ 5.,  5.,  5.]])
>>> a = np.reshape(a, (1, 6))
>>> tstack([a, a, a])
array([[[ 0.,  0.,  0.],
        [ 1.,  1.,  1.],
        [ 2.,  2.,  2.],
        [ 3.,  3.,  3.],
        [ 4.,  4.,  4.],
        [ 5.,  5.,  5.]]])
>>> a = np.reshape(a, (1, 1, 6))
>>> tstack([a, a, a])
array([[[[ 0.,  0.,  0.],
         [ 1.,  1.,  1.],
         [ 2.,  2.,  2.],
         [ 3.,  3.,  3.],
         [ 4.,  4.,  4.],
         [ 5.,  5.,  5.]]]])
```

colour.utilities.tsplit

colour.utilities.**tsplit**(*a*, *dtype=<class 'numpy.float64'>*)

Splits arrays in sequence along the last axis (tail).

Parameters

- **a** (array_like) – Array to perform the splitting.
- **dtype** (object) – Type to use for initial conversion to *ndarray*.

Returns

Return type ndarray

Examples

```
>>> a = np.array([0, 0, 0])
>>> tsplit(a)
array([ 0.,  0.,  0.])
>>> a = np.array(
...     [[0, 0, 0],
...      [1, 1, 1],
...      [2, 2, 2],
...      [3, 3, 3],
...      [4, 4, 4],
...      [5, 5, 5]]
... )
>>> tsplit(a)
array([[ 0.,  1.,  2.,  3.,  4.,  5.],
```

(continues on next page)

(continued from previous page)

```
       [ 0.,  1.,  2.,  3.,  4.,  5.],
       [ 0.,  1.,  2.,  3.,  4.,  5.]])
>>> a = np.array(
...     [[[0, 0, 0],
...       [1, 1, 1],
...       [2, 2, 2],
...       [3, 3, 3],
...       [4, 4, 4],
...       [5, 5, 5]]]
... )
>>> tsplit(a)
array([[[ 0.,  1.,  2.,  3.,  4.,  5.]],

       [[ 0.,  1.,  2.,  3.,  4.,  5.]],

       [[ 0.,  1.,  2.,  3.,  4.,  5.]]])
```

colour.utilities.row_as_diagonal

colour.utilities.**row_as_diagonal**(*a*)

Returns the per row diagonal matrices of the given array.

Parameters **a** (array_like) – Array to perform the diagonal matrices computation.

Returns

Return type ndarray

References

[Cas14]

Examples

```
>>> a = np.array(
...     [[0.25891593, 0.07299478, 0.36586996],
...       [0.30851087, 0.37131459, 0.16274825],
...       [0.71061831, 0.67718718, 0.09562581],
...       [0.71588836, 0.76772047, 0.15476079],
...       [0.92985142, 0.22263399, 0.88027331]]
... )
>>> row_as_diagonal(a)
array([[[ 0.25891593,  0.        ,  0.        ],
        [ 0.        ,  0.07299478,  0.        ],
        [ 0.        ,  0.        ,  0.36586996]],

       [[ 0.30851087,  0.        ,  0.        ],
        [ 0.        ,  0.37131459,  0.        ],
        [ 0.        ,  0.        ,  0.16274825]],

       [[ 0.71061831,  0.        ,  0.        ],
        [ 0.        ,  0.67718718,  0.        ],
        [ 0.        ,  0.        ,  0.09562581]],

       [[ 0.71588836,  0.        ,  0.        ],
        [ 0.        ,  0.76772047,  0.        ],
        [ 0.        ,  0.        ,  0.15476079]],
```

(continues on next page)

(continued from previous page)

```
       [[ 0.92985142,  0.        ,  0.        ],
        [ 0.        ,  0.22263399,  0.        ],
        [ 0.        ,  0.        ,  0.88027331]]])
```

colour.utilities.dot_vector

colour.utilities.**dot_vector**(*m*, *v*)

Convenient wrapper around `np.einsum()` with the following subscripts: '*...ij,...j->...i*'.

It performs the dot product of two arrays where *m* parameter is expected to be an array of 3x3 matrices and parameter *v* an array of vectors.

Parameters

- **m** (array_like) – Array of 3x3 matrices.
- **v** (array_like) – Array of vectors.

Returns

Return type ndarray

Examples

```
>>> m = np.array(
...     [[0.7328, 0.4296, -0.1624],
...      [-0.7036, 1.6975, 0.0061],
...      [0.0030, 0.0136, 0.9834]]
... )
>>> m = np.reshape(np.tile(m, (6, 1)), (6, 3, 3))
>>> v = np.array([0.20654008, 0.12197225, 0.05136952])
>>> v = np.tile(v, (6, 1))
>>> dot_vector(m, v)
array([[ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...],
       [ 0.1954094...,  0.0620396...,  0.0527952...]])
```

colour.utilities.dot_matrix

colour.utilities.**dot_matrix**(*a*, *b*)

Convenient wrapper around `np.einsum()` with the following subscripts: '*...ij,...jk->...ik*'.

It performs the dot product of two arrays where *a* parameter is expected to be an array of 3x3 matrices and parameter *b* another array of of 3x3 matrices.

Parameters

- **a** (array_like) – Array of 3x3 matrices.
- **b** (array_like) – Array of 3x3 matrices.

Returns

Return type ndarray

Examples

```
>>> a = np.array(
...     [[0.7328, 0.4296, -0.1624],
...      [-0.7036, 1.6975, 0.0061],
...      [0.0030, 0.0136, 0.9834]]
... )
>>> a = np.reshape(np.tile(a, (6, 1)), (6, 3, 3))
>>> b = a
>>> dot_matrix(a, b)
array([[[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]],

       [[ 0.2342420...,  1.0418482..., -0.2760903...],
        [-1.7099407...,  2.5793226...,  0.1306181...],
        [-0.0044203...,  0.0377490...,  0.9666713...]]])
```

colour.utilities.orient

colour.utilities.**orient**(*a*, *orientation*)

Orient given array according to given `orientation` value.

Parameters

- **a** (`array_like`) – Array to perform the orientation onto.
- **orientation** (`unicode, optional`) – **{'Flip', 'Flop', '90 CW', '90 CCW', '180'}** Orientation to perform.

Returns Oriented array.

Return type ndarray

Examples

```
>>> a = np.tile(np.arange(5), (5, 1))
>>> a
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
>>> orient(a, '90 CW')
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])
>>> orient(a, 'Flip')
array([[4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0],
       [4, 3, 2, 1, 0]])
```

colour.utilities.centroid

colour.utilities.**centroid**(a)

Computes the centroid indexes of given a array.

Parameters **a** (array_like) – a array to compute the centroid indexes.

Returns a array centroid indexes.

Return type ndarray

Examples

```
>>> a = np.tile(np.arange(0, 5), (5, 1))
>>> centroid(a)
array([2, 3])
```

colour.utilities.linear_conversion

colour.utilities.**linear_conversion**(a, *old_range*, *new_range*)

Performs a simple linear conversion of given array between the old and new ranges.

Parameters

- **a** (array_like) – Array to perform the linear conversion onto.
- **old_range** (array_like) – Old range.
- **new_range** (array_like) – New range.

Returns Linear conversion result.

Return type ndarray

Examples

```
>>> a = np.linspace(0, 1, 10)
>>> linear_conversion(a, np.array([0, 1]), np.array([1, 10]))
array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.])
```

colour.utilities.lerp

colour.utilities.**lerp**(*a*, *b*, *c*)

Performs a simple linear interpolation between given array a and array b using c value.

Parameters

- **a** (array_like) – Array a, the start of the range in which to interpolate.
- **b** (array_like) – Array b, the end of the range in which to interpolate.
- **c** (array_like) – Array c value to use to interpolate between array a and array b.

Returns Linear interpolation result.

Return type ndarray

Examples

```
>>> a = 0
>>> b = 2
>>> lerp(a, b, 0.5)
1.0
```

colour.utilities.fill_nan

colour.utilities.**fill_nan**(*a*, *method='Interpolation'*, *default=0*)

Fills given array NaNs according to given method.

Parameters

- **a** (array_like) – Array to fill the NaNs of.
- **method** (unicode) – **{'Interpolation', 'Constant'}**, *Interpolation* method linearly interpolates through the NaNs, *Constant* method replaces NaNs with `default`.
- **default** (numeric) – Value to use with the *Constant* method.

Returns NaNs filled array.

Return type ndarray

Examples

```
>>> a = np.array([0.1, 0.2, np.nan, 0.4, 0.5])
>>> fill_nan(a)
array([ 0.1,  0.2,  0.3,  0.4,  0.5])
>>> fill_nan(a, method='Constant')
array([ 0.1,  0.2,  0. ,  0.4,  0.5])
```

colour.utilities.ndarray_write

colour.utilities.**ndarray_write**(*a*)

A context manager setting given array writeable to perform an operation and then read-only.

Parameters **a** (array_like) – Array to perform an operation.

Returns Array.

Return type ndarray

Examples

```
>>> a = np.linspace(0, 1, 10)
>>> a.setflags(write=False)
>>> try:
...     a += 1
... except ValueError:
...     pass
>>> with ndarray_write(a):
...     a +=1
```

Metrics

colour.utilities

metric_mse(a, b)	Computes the mean squared error (MSE) or mean squared deviation (MSD) between given *array_like* a and b variables.
metric_psnr(a, b[, max_a])	Computes the peak signal-to-noise ratio (PSNR) between given *array_like* a and b variables.

colour.utilities.metric_mse

colour.utilities.**metric_mse**(*a*, *b*)

Computes the mean squared error (MSE) or mean squared deviation (MSD) between given *array_like* a and b variables.

Parameters

- **a** (array_like) – a variable.
- **b** (array_like) – b variable.

Returns Mean squared error (MSE).

Return type float

References

[Wik03d]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_mse(a, b)
0.0012714...
```

colour.utilities.metric_psnr

colour.utilities.**metric_psnr**(*a*, *b*, *max_a=1*)

Computes the peak signal-to-noise ratio (PSNR) between given *array_like* a and b variables.

Parameters

- **a** (array_like) – a variable.
- **b** (array_like) – b variable.
- **max_a** (numeric, optional) – Maximum possible pixel value of the a variable.

Returns Peak signal-to-noise ratio (PSNR).

Return type float

References

[Wik04a]

Examples

```
>>> a = np.array([0.48222001, 0.31654775, 0.22070353])
>>> b = a * 0.9
>>> metric_psnr(a, b)
28.9568515...
```

Data Structures

colour.utilities

CaseInsensitiveMapping([data])	Implements a case-insensitive mutable mapping / *dict* object.
Lookup	Extends *dict* type to provide a lookup by value(s).
Structure(*args, **kwargs)	Defines a dict-like object allowing to access key values using dot syntax.

colour.utilities.CaseInsensitiveMapping

class colour.utilities.**CaseInsensitiveMapping**(*data=None*, ***kwargs*)

Bases: collections.abc.MutableMapping

Implements a case-insensitive mutable mapping / *dict* object.

Allows values retrieving from keys while ignoring the key case. The keys are expected to be unicode or string-like objects supporting the `str.lower()` method.

Parameters **data** (dict) – *dict* of data to store into the mapping at initialisation.

Other Parameters ****kwargs** (*dict, optional*) – Key / Value pairs to store into the mapping at initialisation.

__setitem__()

__getitem__()

__delitem__()

__contains__()

__iter__()

__len__()

__eq__()

__ne__()

__repr__()

copy()

lower_items()

Warning: The keys are expected to be unicode or string-like objects.

References

[Rei]

Examples

```
>>> methods = CaseInsensitiveMapping({'McCamy': 1, 'Hernandez': 2})
>>> methods['mccamy']
1
```

copy()

Returns a copy of the mapping.

Returns Mapping copy.

Return type *CaseInsensitiveMapping*

Notes

- The colour.utilities.CaseInsensitiveMapping class copy returned is a simple *copy* not a *deepcopy*.

property data
Getter and setter property for the data.

Parameters **value** (dict) – Value to set the data with.

Returns Data.

Return type dict

lower_items()
Iterates over the lower items names.

Returns Lower item names.

Return type generator

colour.utilities.Lookup

class colour.utilities.**Lookup**
Bases: dict

Extends *dict* type to provide a lookup by value(s).

keys_from_value()

first_key_from_value()

References

[Mana]

Examples

```
>>> person = Lookup(first_name='John', last_name='Doe', gender='male')
>>> person.first_key_from_value('John')
'first_name'
>>> persons = Lookup(John='Doe', Jane='Doe', Luke='Skywalker')
>>> sorted(persons.keys_from_value('Doe'))
['Jane', 'John']
```

first_key_from_value(*value*)
Gets the first key with given value.

Parameters **value** (object) – Value.

Returns Key.

Return type object

keys_from_value(*value*)
Gets the keys with given value.

Parameters **value** (object) – Value.

Returns Keys.

Return type object

colour.utilities.Structure

class colour.utilities.**Structure**(**args*, ***kwargs*)

Bases: dict

Defines a dict-like object allowing to access key values using dot syntax.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Key / Value pairs.

References

[Manb]

Examples

```
>>> person = Structure(first_name='John', last_name='Doe', gender='male')
>>> # Doctests skip for Python 2.x compatibility.
>>> person.first_name
'John'
>>> sorted(person.keys())
['first_name', 'gender', 'last_name']
>>> # Doctests skip for Python 2.x compatibility.
>>> person['gender']
'male'
```

Verbose

colour.utilities

message_box(message[, width, padding, ...])	Prints a message inside a box.
warning(*args, **kwargs)	Issues a warning.
filter_warnings([state, colour_warnings, ...])	Filters *Colour* and also optionally overall Python warnings.
suppress_warnings([colour_warnings, ...])	A context manager filtering *Colour* and also optionally overall Python warnings.
numpy_print_options(*args, **kwargs)	A context manager implementing context changes to *Numpy* print behaviour.
describe_environment([runtime_packages, ...])	Describes *Colour* running environment, i.e.

colour.utilities.message_box

colour.utilities.**message_box**(*message*, *width=79*, *padding=3*, *print_callable=<built-in function print>*)

Prints a message inside a box.

Parameters

- **message** (unicode) – Message to print.
- **width** (int, optional) – Message box width.
- **padding** (unicode, optional) – Padding on each sides of the message.

- **print_callable** (callable, optional) – Callable used to print the message box.

Returns Definition success.

Return type bool

Examples

```
>>> message = ('Lorem ipsum dolor sit amet, consectetur adipiscing elit, '
...     'sed do eiusmod tempor incididunt ut labore et dolore magna '
...     'aliqua.')
>>> message_box(message, width=75)
===========================================================================
*                                                                         *
*   Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do       *
*   eiusmod tempor incididunt ut labore et dolore magna aliqua.           *
*                                                                         *
===========================================================================
True
>>> message_box(message, width=60)
============================================================
*                                                          *
*   Lorem ipsum dolor sit amet, consectetur adipiscing     *
*   elit, sed do eiusmod tempor incididunt ut labore et    *
*   dolore magna aliqua.                                   *
*                                                          *
============================================================
True
>>> message_box(message, width=75, padding=16)
===========================================================================
*                                                                         *
*                Lorem ipsum dolor sit amet, consectetur                  *
*                adipiscing elit, sed do eiusmod tempor                   *
*                incididunt ut labore et dolore magna                     *
*                aliqua.                                                  *
*                                                                         *
===========================================================================
True
```

colour.utilities.warning

colour.utilities.**warning**(**args*, ***kwargs*)

Issues a warning.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Keywords arguments.

Returns Definition success.

Return type bool

Examples

```
>>> warning('This is a warning!')
```

colour.utilities.filter_warnings

colour.utilities.**filter_warnings**(*state=True*, *colour_warnings=True*, *colour_runtime_warnings=False*, *colour_usage_warnings=False*, *python_warnings=False*)

Filters *Colour* and also optionally overall Python warnings.

Parameters

- **state** (bool, optional) – Warnings filter state.
- **colour_warnings** (bool, optional) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings.
- **colour_runtime_warnings** (bool, optional) – Whether to filter *Colour* runtime warnings.
- **colour_usage_warnings** (bool, optional) – Whether to filter *Colour* usage warnings.
- **python_warnings** (bool, optional) – Whether to filter *Python* warnings.

Returns Definition success.

Return type bool

Examples

Filtering *Colour* warnings: >>> filter_warnings() True

Filtering *Colour* runtime warnings: >>> filter_warnings(colour_warnings=False, colour_runtime_warnings=True) True

Filtering *Colour* usage warnings: >>> filter_warnings(colour_warnings=False, colour_usage_warnings=True) True

Filtering *Colour* and also Python warnings: >>> filter_warnings(python_warnings=True) True

colour.utilities.suppress_warnings

colour.utilities.**suppress_warnings**(*colour_warnings=True*, *colour_runtime_warnings=False*, *colour_usage_warnings=False*, *python_warnings=False*)

A context manager filtering *Colour* and also optionally overall Python warnings.

Parameters

- **colour_warnings** (bool, optional) – Whether to filter *Colour* warnings, this also filters *Colour* usage and runtime warnings.
- **colour_runtime_warnings** (bool, optional) – Whether to filter *Colour* runtime warnings.
- **colour_usage_warnings** (bool, optional) – Whether to filter *Colour* usage warnings.
- **python_warnings** (bool, optional) – Whether to filter *Python* warnings.

colour.utilities.numpy_print_options

colour.utilities.**numpy_print_options**(**args*, ***kwargs*)

A context manager implementing context changes to *Numpy* print behaviour.

Other Parameters

- ***args** (*list, optional*) – Arguments.
- ****kwargs** (*dict, optional*) – Keywords arguments.

Examples

```
>>> np.array([np.pi])
array([ 3.1415926...])
>>> with numpy_print_options(formatter={'float': '{:0.1f}'.format}):
...     np.array([np.pi])
array([3.1])
```

colour.utilities.describe_environment

colour.utilities.**describe_environment**(*runtime_packages=True*, *development_packages=False*, *extras_packages=False*, *print_environment=True*, ***kwargs*)

Describes *Colour* running environment, i.e. interpreter, runtime and development packages.

Parameters

- **runtime_packages** (bool, optional) – Whether to return the runtime packages versions.
- **development_packages** (bool, optional) – Whether to return the development packages versions.
- **extras_packages** (bool, optional) – Whether to return the extras packages versions.
- **print_environment** (bool, optional) – Whether to print the environment.

Other Parameters

- **padding** (*unicode, optional*) – {colour.utilities.message_box()}, Padding on each sides of the message.
- **print_callable** (*callable, optional*) – {colour.utilities.message_box()}, Callable used to print the message box.
- **width** (*int, optional*) – {colour.utilities.message_box()}, Message box width.

Returns Environment.

Return type defaultdict

Examples

```
>>> environment = describe_environment(width=75)
==============================================================================
*                                                                            *
*   Interpreter :                                                            *
*       python : 3.7.4 (default, Sep  7 2019, 18:27:02)                      *
*                [Clang 10.0.1 (clang-1001.0.46.4)]                          *
*                                                                            *
*   colour-science.org :                                                     *
*       colour : v0.3.13-293-gecf1dc8a                                       *
*                                                                            *
*   Runtime :                                                                *
*       imageio : 2.6.1                                                      *
*       numpy : 1.17.2                                                       *
*       scipy : 1.3.1                                                        *
*       six : 1.12.0                                                         *
*       pandas : 0.24.2                                                      *
*       matplotlib : 3.0.3                                                   *
*       networkx : 2.3                                                       *
*       pygraphviz : 1.5                                                     *
*                                                                            *
==============================================================================
>>> environment = describe_environment(True, True, True, width=75)
...
==============================================================================
*                                                                            *
*   Interpreter :                                                            *
*       python : 3.7.4 (default, Sep  7 2019, 18:27:02)                      *
*                [Clang 10.0.1 (clang-1001.0.46.4)]                          *
*                                                                            *
*   colour-science.org :                                                     *
*       colour : v0.3.13-293-gecf1dc8a                                       *
*                                                                            *
*   Runtime :                                                                *
*       imageio : 2.6.1                                                      *
*       numpy : 1.17.2                                                       *
*       scipy : 1.3.1                                                        *
*       six : 1.12.0                                                         *
*       pandas : 0.24.2                                                      *
*       matplotlib : 3.0.3                                                   *
*       networkx : 2.3                                                       *
*       pygraphviz : 1.5                                                     *
*                                                                            *
*   Development :                                                            *
*       biblib-simple : 0.1.1                                                *
*       coverage : 4.5.4                                                     *
*       coveralls : 1.8.2                                                    *
*       flake8 : 3.7.8                                                       *
*       invoke : 1.3.0                                                       *
*       jupyter : 1.0.0                                                      *
*       mock : 3.0.5                                                         *
*       nose : 1.3.7                                                         *
*       pre-commit : 1.18.3                                                  *
*       pytest : 5.2.1                                                       *
*       restructuredtext-lint : 1.3.0                                        *
*       sphinx : 2.2.0                                                       *
*       sphinx_rtd_theme : 0.4.3                                             *
*       sphinxcontrib-bibtex : 1.0.0                                         *
*       toml : 0.10.0                                                        *
*       twine : 1.15.0                                                       *
*       yapf : 0.23.0                                                        *
```

(continues on next page)

(continued from previous page)

```
*                                                                      *
*   Extras :                                                           *
*       ipywidgets : 7.5.1                                             *
*       notebook : 6.0.1                                               *
*                                                                      *
========================================================================
```

Ancillary Objects

`colour.utilities`

`ColourWarning`	This is the base class of *Colour* warnings.
`ColourUsageWarning`	This is the base class of *Colour* usage warnings.
`ColourRuntimeWarning`	This is the base class of *Colour* runtime warnings.

colour.utilities.ColourWarning

class colour.utilities.**ColourWarning**
Bases: `Warning`

This is the base class of *Colour* warnings. It is a subclass of `Warning` class.

colour.utilities.ColourUsageWarning

class colour.utilities.**ColourUsageWarning**
Bases: `colour.utilities.verbose.ColourWarning`

This is the base class of *Colour* usage warnings. It is a subclass of `colour.utilities.ColourWarning` class.

colour.utilities.ColourRuntimeWarning

class colour.utilities.**ColourRuntimeWarning**
Bases: `colour.utilities.verbose.ColourWarning`

This is the base class of *Colour* runtime warnings. It is a subclass of `colour.utilities.ColourWarning` class.

Colour Volume

- *Optimal Colour Stimuli - MacAdam Limits*
- *Mesh Volume*
- *Pointer's Gamut*
- *RGB Volume*
- *Visible Spectrum*

Optimal Colour Stimuli - MacAdam Limits

colour

is_within_macadam_limits(xyY, illuminant[, ...])	Returns if given *CIE xyY* colourspace array is within MacAdam limits of given illuminant.
ILLUMINANTS_OPTIMAL_COLOUR_STIMULI	Illuminants *Optimal Colour Stimuli*.

colour.is_within_macadam_limits

colour.**is_within_macadam_limits**(*xyY*, *illuminant*, *tolerance=None*)

Returns if given *CIE xyY* colourspace array is within MacAdam limits of given illuminant.

Parameters

- **xyY** (array_like) – *CIE xyY* colourspace array.
- **illuminant** (unicode) – Illuminant.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within MacAdam limits.

Return type bool

Notes

Domain	Scale - Reference	Scale - 1
xyY	[0, 1]	[0, 1]

Examples

```
>>> is_within_macadam_limits(np.array([0.3205, 0.4131, 0.51]), 'A')
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51],
...               [0.0005, 0.0031, 0.001]])
>>> is_within_macadam_limits(a, 'A')
array([ True, False], dtype=bool)
```

colour.ILLUMINANTS_OPTIMAL_COLOUR_STIMULI

colour.**ILLUMINANTS_OPTIMAL_COLOUR_STIMULI = CaseInsensitiveMapping({'A': ..., 'C': ..., 'D65': ...})**

Illuminants *Optimal Colour Stimuli*.

References

[Wik04b]

ILLUMINANTS_OPTIMAL_COLOUR_STIMULI [CaseInsensitiveMapping] **{'A', 'C', 'D65'}**

Mesh Volume

colour

is_within_mesh_volume(points, mesh[, tolerance])	Returns if given points are within given mesh volume using Delaunay triangulation.

colour.is_within_mesh_volume

colour.**is_within_mesh_volume**(*points*, *mesh*, *tolerance=None*)

Returns if given points are within given mesh volume using Delaunay triangulation.

Parameters

- **points** (array_like) – Points to check if they are within mesh volume.
- **mesh** (array_like) – Points of the volume used to generate the Delaunay triangulation.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within mesh volume.

Return type bool

Examples

```
>>> mesh = np.array(
...     [[-1.0, -1.0, 1.0],
...       [1.0, -1.0, 1.0],
...       [1.0, -1.0, -1.0],
...       [-1.0, -1.0, -1.0],
...       [0.0, 1.0, 0.0]]
... )
>>> is_within_mesh_volume(np.array([0.0005, 0.0031, 0.0010]), mesh)
array(True, dtype=bool)
>>> a = np.array([[0.0005, 0.0031, 0.0010],
...               [0.3205, 0.4131, 0.5100]])
>>> is_within_mesh_volume(a, mesh)
array([ True, False], dtype=bool)
```

Pointer's Gamut

colour

is_within_pointer_gamut(XYZ[, tolerance])	Returns if given *CIE XYZ* tristimulus values are within Pointer's Gamut volume.

colour.is_within_pointer_gamut

colour.**is_within_pointer_gamut**(*XYZ*, *tolerance=None*)

Returns if given *CIE XYZ* tristimulus values are within Pointer's Gamut volume.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Returns Is within Pointer's Gamut.

Return type bool

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> is_within_pointer_gamut(np.array([0.3205, 0.4131, 0.5100]))
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.5100], [0.0005, 0.0031, 0.0010]])
>>> is_within_pointer_gamut(a)
array([ True, False], dtype=bool)
```

RGB Volume

colour

RGB_colourspace_limits(colourspace[, illuminant])	Computes given *RGB* colourspace volume limits in *CIE L*a*b** colourspace.
RGB_colourspace_pointer_gamut_coverage_MonteCarlo(...)	Returns given *RGB* colourspace percentage coverage of Pointer's Gamut volume using *Monte Carlo* method.
RGB_colourspace_visible_spectrum_coverage_MonteCarlo(...)	Returns given *RGB* colourspace percentage coverage of visible spectrum volume using *Monte Carlo* method.
RGB_colourspace_volume_MonteCarlo(colourspace)	Performs given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.
RGB_colourspace_volume_coverage_MonteCarlo(...)	Returns given *RGB* colourspace percentage coverage of an arbitrary volume.

colour.RGB_colourspace_limits

colour.**RGB_colourspace_limits**(*colourspace*, *illuminant=array([0.3127, 0.329])*)

Computes given *RGB* colourspace volume limits in *CIE L*a*b** colourspace.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the volume of.
- **illuminant** (array_like, optional) – *CIE L*a*b** colourspace *illuminant* chromaticity coordinates.

Returns *RGB* colourspace volume limits.

Return type ndarray

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> RGB_colourspace_limits(sRGB)
array([[   0.       ...,  100.       ...],
       [ -86.182855 ...,   98.2563272...],
       [-107.8503557...,   94.4894974...]])
```

colour.RGB_colourspace_pointer_gamut_coverage_MonteCarlo

colour.**RGB_colourspace_pointer_gamut_coverage_MonteCarlo**(*colourspace*, *samples=10000000.0*, *random_generator=<function random_triplet_generator>*, *random_state=None*)

Returns given *RGB* colourspace percentage coverage of Pointer's Gamut volume using *Monte Carlo* method.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the *Pointer's Gamut* coverage percentage.
- **samples** (numeric, optional) – Samples count.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of *Pointer's Gamut* volume.

Return type float

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_pointer_gamut_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng)
81...
```

colour.RGB_colourspace_visible_spectrum_coverage_MonteCarlo

colour.**RGB_colourspace_visible_spectrum_coverage_MonteCarlo**(*colourspace*, *samples=10000000.0*, *random_generator=<function random_triplet_generator>*, *random_state=None*)

Returns given *RGB* colourspace percentage coverage of visible spectrum volume using *Monte Carlo* method.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the visible spectrum coverage percentage.
- **samples** (numeric, optional) – Samples count.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of visible spectrum volume.

Return type float

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_visible_spectrum_coverage_MonteCarlo(
...     sRGB, 10e3, random_state=prng)
46...
```

colour.RGB_colourspace_volume_MonteCarlo

colour.**RGB_colourspace_volume_MonteCarlo**(*colourspace*, *samples=10000000.0*, *limits=array([[0., 100.], [-150., 150.], [-150., 150.]])*, *illuminant_Lab=array([0.3127, 0.329])*, *chromatic_adaptation_method='CAT02'*, *random_generator=<function random_triplet_generator>*, *random_state=None*)

Performs given *RGB* colourspace volume computation using *Monte Carlo* method and multiprocessing.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the volume of.
- **samples** (numeric, optional) – Samples count.

- **limits** (array_like, optional) – *CIE L*a*b** colourspace volume.
- **illuminant_Lab** (array_like, optional) – *CIE L*a*b** colourspace *illuminant* chromaticity coordinates.
- **chromatic_adaptation_method** (unicode, optional) – **{'CAT02', 'XYZ Scaling', 'Von Kries', 'Bradford', 'Sharp', 'Fairchild', 'CMCCAT97', 'CMCCAT2000', 'CAT02_BRILL_CAT', 'Bianco', 'Bianco PC'}**, *Chromatic adaptation* method.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples within the *CIE L*a*b** colourspace volume.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns *RGB* colourspace volume.

Return type float

Notes

- The doctest is assuming that `np.random.RandomState()` definition will return the same sequence no matter which *OS* or *Python* version is used. There is however no formal promise about the *prng* sequence reproducibility of either *Python* or *Numpy* implementations: Laurent. (2012). Reproducibility of python pseudo-random numbers across systems and versions? Retrieved January 20, 2015, from http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> from colour.utilities import disable_multiprocessing
>>> prng = np.random.RandomState(2)
>>> with disable_multiprocessing():
...     RGB_colourspace_volume_MonteCarlo(sRGB, 10e3, random_state=prng)
...
8...
```

colour.RGB_colourspace_volume_coverage_MonteCarlo

colour.**RGB_colourspace_volume_coverage_MonteCarlo**(*colourspace*, *coverage_sampler*, *samples=10000000.0*, *random_generator=<function random_triplet_generator>*, *random_state=None*)

Returns given *RGB* colourspace percentage coverage of an arbitrary volume.

Parameters

- **colourspace** (RGB_Colourspace) – *RGB* colourspace to compute the volume coverage percentage.
- **coverage_sampler** (object) – Python object responsible for checking the volume coverage.
- **samples** (numeric, optional) – Samples count.
- **random_generator** (generator, optional) – Random triplet generator providing the random samples.
- **random_state** (RandomState, optional) – Mersenne Twister pseudo-random number generator to use in the random number generator.

Returns Percentage coverage of volume.

Return type float

Examples

```
>>> from colour.models import sRGB_COLOURSPACE as sRGB
>>> prng = np.random.RandomState(2)
>>> RGB_colourspace_volume_coverage_MonteCarlo(
...     sRGB, is_within_pointer_gamut, 10e3, random_state=prng)
...
81...
```

Visible Spectrum

colour

is_within_visible_spectrum(XYZ[, cmfs, ...])	Returns if given *CIE XYZ* tristimulus values are within visible spectrum volume / given colour matching functions volume.

colour.is_within_visible_spectrum

colour.**is_within_visible_spectrum**(*XYZ*, *cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, *tolerance=None*, ***kwargs*)

Returns if given *CIE XYZ* tristimulus values are within visible spectrum volume / given colour matching functions volume.

Parameters

- **XYZ** (array_like) – *CIE XYZ* tristimulus values.
- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.
- **tolerance** (numeric, optional) – Tolerance allowed in the inside-triangle check.

Other Parameters ****kwargs** (*dict, optional*) – {colour.multi_sds_to_XYZ()}, Please refer to the documentation of the previously listed definition.

Returns Is within visible spectrum.

Return type bool

Notes

Domain	Scale - Reference	Scale - 1
XYZ	[0, 1]	[0, 1]

Examples

```
>>> import numpy as np
>>> is_within_visible_spectrum(np.array([0.3205, 0.4131, 0.51]))
array(True, dtype=bool)
>>> a = np.array([[0.3205, 0.4131, 0.51],
...               [-0.0005, 0.0031, 0.001]])
>>> is_within_visible_spectrum(a)
array([ True, False], dtype=bool)
```

Ancillary Objects

colour.volume

generate_pulse_waves(bins)	Generates the pulse waves of given number of bins necessary to totally stimulate the colour matching functions.
XYZ_outer_surface([cmfs, illuminant])	Generates the *CIE XYZ* colourspace outer surface for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.

colour.volume.generate_pulse_waves

colour.volume.**generate_pulse_waves**(*bins*)

Generates the pulse waves of given number of bins necessary to totally stimulate the colour matching functions.

Assuming 5 bins, a first set of SPDs would be as follows:

```
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

The second one:

```
1 1 0 0 0
0 1 1 0 0
0 0 1 1 0
0 0 0 1 1
1 0 0 0 1
```

The third:

1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1

Etc. . .

Parameters **bins** (int) – Number of bins of the pulse waves.

Returns Pulse waves.

Return type ndarray

References

[Lin15], [Man18]

Examples

```
>>> generate_pulse_waves(5)
array([[ 0.,  0.,  0.,  0.,  0.],
       [ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.],
       [ 1.,  1.,  0.,  0.,  0.],
       [ 0.,  1.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  1.,  0.],
       [ 0.,  0.,  0.,  1.,  1.],
       [ 1.,  0.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  0.,  0.],
       [ 0.,  1.,  1.,  1.,  0.],
       [ 0.,  0.,  1.,  1.,  1.],
       [ 1.,  0.,  0.,  1.,  1.],
       [ 1.,  1.,  0.,  0.,  1.],
       [ 1.,  1.,  1.,  1.,  0.],
       [ 0.,  1.,  1.,  1.,  1.],
       [ 1.,  0.,  1.,  1.,  1.],
       [ 1.,  1.,  0.,  1.,  1.],
       [ 1.,  1.,  1.,  0.,  1.],
       [ 1.,  1.,  1.,  1.,  1.]])
```

colour.volume.XYZ_outer_surface

colour.volume.**XYZ_outer_surface**(*cmfs=XYZ_ColourMatchingFunctions(name='CIE 1931 2 Degree Standard Observer', ...)*, *illuminant=SpectralDistribution(name='1 Constant', ...)*, ***kwargs*)

Generates the *CIE XYZ* colourspace outer surface for given colour matching functions using multi-spectral conversion of pulse waves to *CIE XYZ* tristimulus values.

Parameters

- **cmfs** (XYZ_ColourMatchingFunctions, optional) – Standard observer colour matching functions.
- **illuminant** (SpectralDistribution, optional) – Illuminant spectral distribution.

Other Parameters ****kwargs** (*dict, optional*) – {colour.multi_sds_to_XYZ()}, Please refer to the documentation of the previously listed definition.

Returns Outer surface *CIE XYZ* tristimulus values.

Return type ndarray

References

[Lin15], [Man18]

Examples

```
>>> from colour.colorimetry import DEFAULT_SPECTRAL_SHAPE
>>> shape = SpectralShape(
...     DEFAULT_SPECTRAL_SHAPE.start, DEFAULT_SPECTRAL_SHAPE.end, 84)
>>> cmfs = STANDARD_OBSERVERS_CMFS['CIE 1931 2 Degree Standard Observer']
>>> XYZ_outer_surface(cmfs.copy().align(shape))
array([[  0.0000000...e+00,   0.0000000...e+00,   0.0000000...e+00],
       [  9.6361381...e-05,   2.9056776...e-06,   4.4961226...e-04],
       [  2.5910529...e-01,   2.1031298...e-02,   1.3207468...e+00],
       [  1.0561021...e-01,   6.2038243...e-01,   3.5423571...e-02],
       [  7.2647980...e-01,   3.5460869...e-01,   2.1005149...e-04],
       [  1.0971874...e-02,   3.9635453...e-03,   0.0000000...e+00],
       [  3.0792572...e-05,   1.1119762...e-05,   0.0000000...e+00],
       [  2.5920165...e-01,   2.1034203...e-02,   1.3211965...e+00],
       [  3.6471551...e-01,   6.4141373...e-01,   1.3561704...e+00],
       [  8.3209002...e-01,   9.7499113...e-01,   3.5633622...e-02],
       [  7.3745167...e-01,   3.5857224...e-01,   2.1005149...e-04],
       [  1.1002667...e-02,   3.9746651...e-03,   0.0000000...e+00],
       [  1.2715395...e-04,   1.4025439...e-05,   4.4961226...e-04],
       [  3.6481187...e-01,   6.4141663...e-01,   1.3566200...e+00],
       [  1.0911953...e+00,   9.9602242...e-01,   1.3563805...e+00],
       [  8.4306189...e-01,   9.7895467...e-01,   3.5633622...e-02],
       [  7.3748247...e-01,   3.5858336...e-01,   2.1005149...e-04],
       [  1.1099028...e-02,   3.9775708...e-03,   4.4961226...e-04],
       [  2.5923244...e-01,   2.1045323...e-02,   1.3211965...e+00],
       [  1.0912916...e+00,   9.9602533...e-01,   1.3568301...e+00],
       [  1.1021671...e+00,   9.9998597...e-01,   1.3563805...e+00],
       [  8.4309268...e-01,   9.7896579...e-01,   3.5633622...e-02],
       [  7.3757883...e-01,   3.5858626...e-01,   6.5966375...e-04],
       [  2.7020432...e-01,   2.5008868...e-02,   1.3211965...e+00],
       [  3.6484266...e-01,   6.4142775...e-01,   1.3566200...e+00],
       [  1.1022635...e+00,   9.9998888...e-01,   1.3568301...e+00],
       [  1.1021979...e+00,   9.9999709...e-01,   1.3563805...e+00],
       [  8.4318905...e-01,   9.7896870...e-01,   3.6083235...e-02],
       [  9.9668412...e-01,   3.7961756...e-01,   1.3214065...e+00],
       [  3.7581454...e-01,   6.4539130...e-01,   1.3566200...e+00],
       [  1.0913224...e+00,   9.9603645...e-01,   1.3568301...e+00],
       [  1.1022943...e+00,   1.0000000...e+00,   1.3568301...e+00]])
```

Indices and tables

- genindex
- search

4.3.1.5 Bibliography

Indirect References

Some extra references used in the codebase but not directly part of the public api:

- [Cen14e]
- [Cen14k]
- [Cen14h]
- [Cen14c]
- [Cen14j]
- [Cen14i]
- [Cen14g]
- [Cen14d]
- [Cen14f]
- [Cen14b]
- [Cen14a]
- [CIET13805d]
- [Dji17]
- [FiLMiCInc17]
- [Hou15]
- [Lau12]
- [Mac35]
- [Mac42]
- [MunsellCScienceb]
- [Poi80]
- [RenewableRDCenter03]
- [SWD05]
- [Sir18]
- [SHF00]
- [WEV02]
- [War16]
- [WS00b]
- [WS00j]
- [WS00l]
- [WS00h]

4.4 5.4 Jupyter Notebooks

Jupyter Notebooks are available and designed to provide an historical perspective of colour science via **Colour** usage.

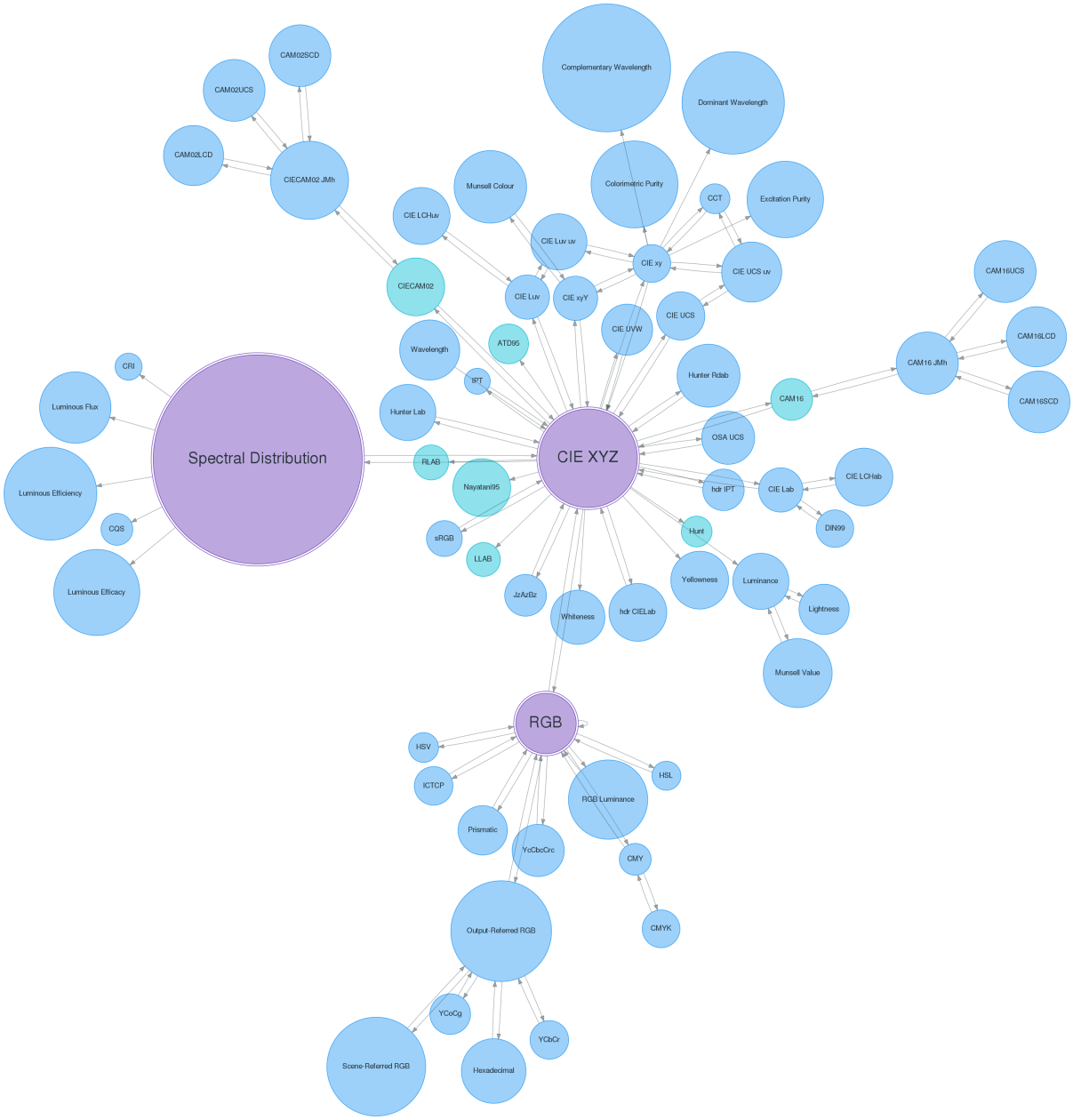
4.5 5.5 Examples

Most of the objects are available from the `colour` namespace:

```
>>> import colour
```

4.5.1 5.5.1 Automatic Colour Conversion Graph - `colour.graph`

Starting with version *0.3.14*, **Colour** implements an automatic colour conversion graph enabling easier colour conversions.



```
>>> sd = colour.COLOURCHECKERS_SDS['ColorChecker N Ohta']['dark skin']
>>> convert(sd, 'Spectral Distribution', 'sRGB', verbose={'mode': 'Short'})
```

```
================================================================================
*                                                                              *
*   [ Conversion Path ]                                                        *
*                                                                              *
*   "sd_to_XYZ" --> "XYZ_to_sRGB"                                              *
*                                                                              *
================================================================================
array([ 0.45675795,  0.30986982,  0.24861924])
```

```
>>> illuminant = colour.ILLUMINANTS_SDS['FL2']
>>> convert(sd, 'Spectral Distribution', 'sRGB', sd_to_XYZ={'illuminant': illuminant})
array([ 0.47924575,  0.31676968,  0.17362725])
```

4.5.2 5.5.2 Chromatic Adaptation - `colour.adaptation`

```
>>> XYZ = [0.20654008, 0.12197225, 0.05136952]
>>> D65 = colour.ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['D65']
>>> A = colour.ILLUMINANTS['CIE 1931 2 Degree Standard Observer']['A']
>>> colour.chromatic_adaptation(
...     XYZ, colour.xy_to_XYZ(D65), colour.xy_to_XYZ(A))
array([ 0.2533053 ,  0.13765138,  0.01543307])
>>> sorted(colour.CHROMATIC_ADAPTATION_METHODS.keys())
['CIE 1994', 'CMCCAT2000', 'Fairchild 1990', 'Von Kries']
```

4.5.3 5.5.3 Algebra - `colour.algebra`

4.5.3.1 5.5.3.1 Kernel Interpolation

```
>>> y = [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
>>> x = range(len(y))
>>> colour.KernelInterpolator(x, y)([0.25, 0.75, 5.50])
array([  6.18062083,   8.08238488,  57.85783403])
```

4.5.3.2 5.5.3.2 Sprague (1880) Interpolation

```
>>> y = [5.9200, 9.3700, 10.8135, 4.5100, 69.5900, 27.8007, 86.0500]
>>> x = range(len(y))
>>> colour.SpragueInterpolator(x, y)([0.25, 0.75, 5.50])
array([  6.72951612,   7.81406251,  43.77379185])
```

4.5.4 5.5.4 Colour Appearance Models - `colour.appearance`

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952 * 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> colour.XYZ_to_CIECAM02(XYZ, XYZ_w, L_A, Y_b)
CIECAM02_Specification(J=34.434525727858997, C=67.365010921125915, h=22.279164147957076, s=62.
↪814855853327131, Q=177.47124941102123, M=70.024939419291385, H=2.689608534423904, HC=None)
```

4.5.5 5.5.5 Colour Blindness - colour.blindness

```
>>> import colour
>>> cmfs = colour.LMS_CMFS['Stockman & Sharpe 2 Degree Cone Fundamentals']
>>> colour.anomalous_trichromacy_cmfs_Machado2009(cmfs, np.array([15, 0, 0]))[450]
array([ 0.08912884,  0.0870524 ,  0.955393  ])
>>> primaries = colour.DISPLAYS_RGB_PRIMARIES['Apple Studio Display']
>>> d_LMS = (15, 0, 0)
>>> colour.anomalous_trichromacy_matrix_Machado2009(cmfs, primaries, d_LMS)
array([[-0.27774652,  2.65150084, -1.37375432],
       [ 0.27189369,  0.20047862,  0.52762768],
       [ 0.00644047,  0.25921579,  0.73434374]])
```

4.5.6 5.5.6 Colour Correction - colour characterisation

```
>>> import numpy as np
>>> RGB = [0.17224810, 0.09170660, 0.06416938]
>>> M_T = np.random.random((24, 3))
>>> M_R = M_T + (np.random.random((24, 3)) - 0.5) * 0.5
>>> colour.colour_correction(RGB, M_T, M_R)
array([ 0.15205429,  0.08974029,  0.04141435])
>>> sorted(colour.COLOUR_CORRECTION_METHODS.keys())
['Cheung 2004', 'Finlayson 2015', 'Vandermonde']
```

4.5.7 5.5.7 Colorimetry - colour.colorimetry

4.5.7.1 5.5.7.1 Spectral Computations

```
>>> colour.sd_to_XYZ(colour.LIGHT_SOURCES_SDS['Neodimium Incandescent'])
array([ 36.94726204,  32.62076174,  13.0143849 ])
>>> sorted(colour.SPECTRAL_TO_XYZ_METHODS.keys())
['ASTM E308', 'Integration', 'astm2015']
```

4.5.7.2 5.5.7.2 Multi-Spectral Computations

```
>>> msds = np.array([
...     [[0.01367208, 0.09127947, 0.01524376, 0.02810712, 0.19176012, 0.04299992],
...      [0.00959792, 0.25822842, 0.41388571, 0.22275120, 0.00407416, 0.37439537],
...      [0.01791409, 0.29707789, 0.56295109, 0.23752193, 0.00236515, 0.58190280]],
...     [[0.01492332, 0.10421912, 0.02240025, 0.03735409, 0.57663846, 0.32416266],
...      [0.04180972, 0.26402685, 0.03572137, 0.00413520, 0.41808194, 0.24696727],
...      [0.00628672, 0.11454948, 0.02198825, 0.39906919, 0.63640803, 0.01139849]],
...     [[0.04325933, 0.26825359, 0.23732357, 0.05175860, 0.01181048, 0.08233768],
...      [0.02484169, 0.12027161, 0.00541695, 0.00654612, 0.18603799, 0.36247808],
...      [0.03102159, 0.16815442, 0.37186235, 0.08610666, 0.00413520, 0.78492409]],
...     [[0.11682307, 0.78883040, 0.74468607, 0.83375293, 0.90571451, 0.70054168],
...      [0.06321812, 0.41898224, 0.15190357, 0.24591440, 0.55301750, 0.00657664],
...      [0.00305180, 0.11288624, 0.11357290, 0.12924391, 0.00195315, 0.21771573]],
... ])
>>> colour.multi_sds_to_XYZ(msds, cmfs, illuminant, method='Integration',
...                         shape=colour.SpectralShape(400, 700, 60)))
[[[  9.73192501   5.02105851   3.22790699]
  [ 16.08032168  24.47303359  10.28681006]
  [ 17.73513774  29.61865582  12.10713449]]
 [[ 25.69298792  11.72611193   3.70187275]
  [ 18.51208526   8.03720984   9.30361825]
```

(continues on next page)

(continued from previous page)

```
  [ 48.55945054  32.30885571   4.09223401]]
 [[  5.7743232   10.10692925  10.08461311]
  [  8.81306527   3.65394599   4.20783881]
  [  8.06007398  15.87077693   7.02551086]]
 [[ 90.88877129  81.82966846  29.86765971]
  [ 38.64801062  26.70860262  15.08396538]
  [  8.77151115  10.56330761   4.28940206]]]
>>> sorted(colour.MULTI_SPECTRAL_TO_XYZ_METHODS.keys())
['ASTM E308', 'Integration', 'astm2015']
```

4.5.7.3 5.5.7.3 Blackbody Spectral Radiance Computation

```
>>> colour.sd_blackbody(5000)
SpectralDistribution([[  3.60000000e+02,   6.65427827e+12],
                      [  3.61000000e+02,   6.70960528e+12],
                      [  3.62000000e+02,   6.76482512e+12],
                      ...
                      [  7.78000000e+02,   1.06068004e+13],
                      [  7.79000000e+02,   1.05903327e+13],
                      [  7.80000000e+02,   1.05738520e+13]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={'right': None, 'method': 'Constant', 'left': None})
```

4.5.7.4 5.5.7.4 Dominant, Complementary Wavelength & Colour Purity Computation

```
>>> xy = [0.54369557, 0.32107944]
>>> xy_n = [0.31270000, 0.32900000]
>>> colour.dominant_wavelength(xy, xy_n)
(array(616.0),
 array([ 0.68354746,  0.31628409]),
 array([ 0.68354746,  0.31628409]))
```

4.5.7.5 5.5.7.5 Lightness Computation

```
>>> colour.lightness(12.19722535)
41.527875844653451
>>> sorted(colour.LIGHTNESS_METHODS.keys())
['CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Glasser 1958',
 'Lstar1976',
 'Wyszecki 1963']
```

4.5.7.6 5.5.7.6 Luminance Computation

```
>>> colour.luminance(41.52787585)
12.197225353400775
>>> sorted(colour.LUMINANCE_METHODS.keys())
['ASTM D1535',
 'CIE 1976',
 'Fairchild 2010',
 'Fairchild 2011',
 'Newhall 1943',
 'astm2008',
 'cie1976']
```

4.5.7.7 5.5.7.7 Whiteness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> XYZ_0 = [94.80966767, 100.00000000, 107.30513595]
>>> colour.whiteness(XYZ, XYZ_0)
array([ 93.756     ,  -1.33000001])
>>> sorted(colour.WHITENESS_METHODS.keys())
['ASTM E313',
 'Berger 1959',
 'CIE 2004',
 'Ganz 1979',
 'Stensby 1968',
 'Taube 1960',
 'cie2004']
```

4.5.7.8 5.5.7.8 Yellowness Computation

```
>>> XYZ = [95.00000000, 100.00000000, 105.00000000]
>>> colour.yellowness(XYZ)
11.065000000000003
>>> sorted(colour.YELLOWNESS_METHODS.keys())
['ASTM D1925', 'ASTM E313']
```

4.5.7.9 5.5.7.9 Luminous Flux, Efficiency & Efficacy Computation

```
>>> sd = colour.LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> colour.luminous_flux(sd)
23807.655527367202
>>> sd = colour.LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> colour.luminous_efficiency(sd)
0.19943935624521045
>>> sd = colour.LIGHT_SOURCES_SDS['Neodimium Incandescent']
>>> colour.luminous_efficacy(sd)
136.21708031547874
```

4.5.8 5.5.8 Contrast Sensitivity Function - `colour.contrast`

```
>>> colour.contrast_sensitivity_function(u=4, X_0=60, E=65)
358.51180789884984
>>> sorted(colour.CONTRAST_SENSITIVITY_METHODS.keys())
['Barten 1999']
```

4.5.9 5.5.9 Colour Difference - `colour.difference`

```
>>> Lab_1 = [100.00000000, 21.57210357, 272.22819350]
>>> Lab_2 = [100.00000000, 426.67945353, 72.39590835]
>>> colour.delta_E(Lab_1, Lab_2)
94.035649026659485
>>> sorted(colour.DELTA_E_METHODS.keys())
['CAM02-LCD',
 'CAM02-SCD',
 'CAM02-UCS',
 'CAM16-LCD',
 'CAM16-SCD',
 'CAM16-UCS',
 'CIE 1976',
 'CIE 1994',
 'CIE 2000',
 'CMC',
 'DIN99',
 'cie1976',
 'cie1994',
 'cie2000']
```

4.5.10 5.5.10 IO - `colour.io`

4.5.10.1 5.5.10.1 Images

```
>>> RGB = colour.read_image('Ishihara_Colour_Blindness_Test_Plate_3.png')
>>> RGB.shape
(276, 281, 3)
```

4.5.10.2 5.5.10.2 Look Up Table (LUT) Data

```
>>> LUT = colour.read_LUT('ACES_Proxy_10_to_ACES.cube')
>>> print(LUT)
```

```
LUT3x1D - ACES Proxy 10 to ACES
-------------------------------
Dimensions : 2
Domain     : [[0 0 0]
              [1 1 1]]
Size       : (32, 3)
```

```
>>> RGB = [0.17224810, 0.09170660, 0.06416938]
>>> LUT.apply(RGB)
array([ 0.00575674,  0.00181493,  0.00121419])
```

4.5.11 5.5.11 Colour Models - `colour.models`

4.5.11.1 5.5.11.1 CIE xyY Colourspace

```
>>> colour.XYZ_to_xyY([0.20654008, 0.12197225, 0.05136952])
array([ 0.54369557,  0.32107944,  0.12197225])
```

4.5.11.2 5.5.11.2 CIE L*a*b* Colourspace

```
>>> colour.XYZ_to_Lab([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  52.63858304,  26.92317922])
```

4.5.11.3 5.5.11.3 CIE L*u*v* Colourspace

```
>>> colour.XYZ_to_Luv([0.20654008, 0.12197225, 0.05136952])
array([ 41.52787529,  96.83626054,  17.75210149])
```

4.5.11.4 5.5.11.4 CIE 1960 UCS Colourspace

```
>>> colour.XYZ_to_UCS([0.20654008, 0.12197225, 0.05136952])
array([ 0.13769339,  0.12197225,  0.1053731 ])
```

4.5.11.5 5.5.11.5 CIE 1964 U*V*W* Colourspace

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952* 100]
>>> colour.XYZ_to_UVW(XYZ)
array([ 94.55035725,  11.55536523,  40.54757405])
```

4.5.11.6 5.5.11.6 Hunter L,a,b Colour Scale

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952* 100]
>>> colour.XYZ_to_Hunter_Lab(XYZ)
array([ 34.92452577,  47.06189858,  14.38615107])
```

4.5.11.7 5.5.11.7 Hunter Rd,a,b Colour Scale

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952* 100]
>>> colour.XYZ_to_Hunter_Rdab(XYZ)
array([ 12.197225  ,  57.12537874,  17.46241341])
```

4.5.11.8 5.5.11.8 CAM02-LCD, CAM02-SCD, and CAM02-UCS Colourspaces - Luo, Cui and Li (2006)

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952* 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = colour.CIECAM02_VIEWING_CONDITIONS['Average']
>>> specification = colour.XYZ_to_CIECAM02(
        XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CIECAM02_to_CAM02UCS(JMh)
array([ 47.16899898,  38.72623785,  15.8663383 ])
```

4.5.11.9 5.5.11.9 CAM16-LCD, CAM16-SCD, and CAM16-UCS Colourspaces - Li et al. (2017)

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952* 100]
>>> XYZ_w = [95.05, 100.00, 108.88]
>>> L_A = 318.31
>>> Y_b = 20.0
>>> surround = colour.CAM16_VIEWING_CONDITIONS['Average']
>>> specification = colour.XYZ_to_CAM16(
        XYZ, XYZ_w, L_A, Y_b, surround)
>>> JMh = (specification.J, specification.M, specification.h)
>>> colour.JMh_CAM16_to_CAM16UCS(JMh)
array([ 46.55542238,  40.22460974,  14.25288392]
```

4.5.11.10 5.5.11.10 IPT Colourspace

```
>>> colour.XYZ_to_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 0.38426191,  0.38487306,  0.18886838])
```

4.5.11.11 5.5.11.11 DIN99 Colourspace

```
>>> Lab = [41.52787529, 52.63858304, 26.92317922]
>>> colour.Lab_to_DIN99(Lab)
array([ 53.22821988,  28.41634656,   3.89839552])
```

4.5.11.12 5.5.11.12 hdr-CIELAB Colourspace

```
>>> colour.XYZ_to_hdr_CIELab([0.20654008, 0.12197225, 0.05136952])
array([ 51.87002062,  60.4763385 ,  32.14551912])
```

4.5.11.13 5.5.11.13 hdr-IPT Colourspace

```
>>> colour.XYZ_to_hdr_IPT([0.20654008, 0.12197225, 0.05136952])
array([ 25.18261761, -22.62111297,   3.18511729])
```

4.5.11.14 5.5.11.14 OSA UCS Colourspace

```
>>> XYZ = [0.20654008 * 100, 0.12197225 * 100, 0.05136952* 100]
>>> colour.XYZ_to_OSA_UCS(XYZ)
array([-3.0049979 ,  2.99713697, -9.66784231])
```

4.5.11.15 5.5.11.15 JzAzBz Colourspace

```
>>> colour.XYZ_to_JzAzBz([0.20654008, 0.12197225, 0.05136952])
array([ 0.00535048,  0.00924302,  0.00526007])
```

4.5.11.16 5.5.11.16 Y'CbCr Colour Encoding

```
>>> colour.RGB_to_YCbCr([1.0, 1.0, 1.0])
array([ 0.92156863,  0.50196078,  0.50196078])
```

4.5.11.17 5.5.11.17 YCoCg Colour Encoding

```
>>> colour.RGB_to_YCoCg([0.75, 0.75, 0.0])
array([ 0.5625,  0.375 ,  0.1875])
```

4.5.11.18 5.5.11.18 ICTCP Colour Encoding

```
>>> colour.RGB_to_ICTCP([0.45620519, 0.03081071, 0.04091952])
array([ 0.07351364,  0.00475253,  0.09351596])
```

4.5.11.19 5.5.11.19 HSV Colourspace

```
>>> colour.RGB_to_HSV([0.45620519, 0.03081071, 0.04091952])
array([ 0.99603944,  0.93246304,  0.45620519])
```

4.5.11.20 5.5.11.20 Prismatic Colourspace

```
>>> colour.RGB_to_Prismatic([0.25, 0.50, 0.75])
array([ 0.75      ,  0.16666667,  0.33333333,  0.5       ])
```

4.5.11.21 5.5.11.21 RGB Colourspace and Transformations

```
>>> XYZ = [0.21638819, 0.12570000, 0.03847493]
>>> illuminant_XYZ = [0.34570, 0.35850]
>>> illuminant_RGB = [0.31270, 0.32900]
>>> chromatic_adaptation_transform = 'Bradford'
>>> XYZ_to_RGB_matrix = [
        [3.24062548, -1.53720797, -0.49862860],
        [-0.96893071, 1.87575606, 0.04151752],
        [0.05571012, -0.20402105, 1.05699594]]
>>> colour.XYZ_to_RGB(
        XYZ,
        illuminant_XYZ,
```

(continues on next page)

(continued from previous page)

```
        illuminant_RGB,
        XYZ_to_RGB_matrix,
        chromatic_adaptation_transform)
array([ 0.45595571,  0.03039702,  0.04087245])
```

4.5.11.22 5.5.11.22 RGB Colourspace Derivation

```
>>> p = [0.73470, 0.26530, 0.00000, 1.00000, 0.00010, -0.07700]
>>> w = [0.32168, 0.33767]
>>> colour.normalised_primary_matrix(p, w)
array([[  9.52552396e-01,   0.00000000e+00,   9.36786317e-05],
       [  3.43966450e-01,   7.28166097e-01,  -7.21325464e-02],
       [  0.00000000e+00,   0.00000000e+00,   1.00882518e+00]])
```

4.5.11.23 5.5.11.23 RGB Colourspaces

```
>>> sorted(colour.RGB_COLOURSPACES.keys())
['ACES2065-1',
 'ACEScc',
 'ACEScct',
 'ACEScg',
 'ACESproxy',
 'ALEXA Wide Gamut',
 'Adobe RGB (1998)',
 'Adobe Wide Gamut RGB',
 'Apple RGB',
 'Best RGB',
 'Beta RGB',
 'CIE RGB',
 'Cinema Gamut',
 'ColorMatch RGB',
 'DCDM XYZ',
 'DCI-P3',
 'DCI-P3+',
 'Display P3',
 'DJI D-Gamut',
 'DRAGONcolor',
 'DRAGONcolor2',
 'Don RGB 4',
 'ECI RGB v2',
 'ERIMM RGB',
 'Ekta Space PS 5',
 'F-Gamut',
 'FilmLight E-Gamut',
 'ITU-R BT.2020',
 'ITU-R BT.470 - 525',
 'ITU-R BT.470 - 625',
 'ITU-R BT.709',
 'Max RGB',
 'NTSC (1953)',
 'NTSC (1987)',
 'P3-D65',
 'Pal/Secam',
 'ProPhoto RGB',
 'Protune Native',
 'REDWideGamutRGB',
 'REDcolor',
```

(continues on next page)

(continued from previous page)

```
 'REDcolor2',
 'REDcolor3',
 'REDcolor4',
 'RIMM RGB',
 'ROMM RGB',
 'Russell RGB',
 'S-Gamut',
 'S-Gamut3',
 'S-Gamut3.Cine',
 'SMPTE 240M',
 'SMPTE C',
 'Sharp RGB',
 'V-Gamut',
 'Xtreme RGB',
 'aces',
 'adobe1998',
 'prophoto',
 'sRGB']
```

4.5.11.24 5.5.11.24 OETFs

```
>>> sorted(colour.OETFS.keys())
['ARIB STD-B67',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'SMPTE 240M']
```

4.5.11.25 5.5.11.25 OETFs Inverse

```
>>> sorted(colour.OETF_INVERSES.keys())
['ARIB STD-B67',
 'ITU-R BT.2100 HLD',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709']
```

4.5.11.26 5.5.11.26 EOTFs

```
>>> sorted(colour.EOTFS.keys())
['DCDM',
 'DICOM GSDF',
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'SMPTE 240M',
 'ST 2084',
 'sRGB']
```

4.5.11.27 5.5.11.27 EOTFs Inverse

```
>>> sorted(colour.EOTF_INVERSES.keys())
['DCDM',
 'DICOM GSDF',
 'ITU-R BT.1886',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ST 2084',
 'sRGB']
```

4.5.11.28 5.5.11.28 OOTFs

```
>>> sorted(colour.OOTFS.keys())
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

4.5.11.29 5.5.11.29 OOTFs Inverse

```
>>> sorted(colour.OOTF_INVERSES.keys())
['ITU-R BT.2100 HLG', 'ITU-R BT.2100 PQ']
```

4.5.11.30 5.5.11.30 Log Encoding / Decoding

```
>>> sorted(colour.LOG_ENCODINGS.keys())
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ALEXA Log C',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'D-Log',
 'ERIMM RGB',
 'F-Log',
 'Filmic Pro 6',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'T-Log',
 'V-Log',
 'ViperLog']
```

4.5.11.31 5.5.11.31 CCTFs Encoding / Decoding

```
>>> sorted(colour.CCTF_ENCODINGS.keys())
['ACEScc',
 'ACEScct',
 'ACESproxy',
 'ALEXA Log C',
 'ARIB STD-B67',
 'Canon Log',
 'Canon Log 2',
 'Canon Log 3',
 'Cineon',
 'D-Log',
 'DCDM',
 'DICOM GSDF',
 'ERIMM RGB',
 'F-Log',
 'Filmic Pro 6',
 'Gamma 2.2',
 'Gamma 2.4',
 'Gamma 2.6',
 'ITU-R BT.1886',
 'ITU-R BT.2020',
 'ITU-R BT.2100 HLG',
 'ITU-R BT.2100 PQ',
 'ITU-R BT.601',
 'ITU-R BT.709',
 'Log3G10',
 'Log3G12',
 'PLog',
 'Panalog',
 'ProPhoto RGB',
 'Protune',
 'REDLog',
 'REDLogFilm',
 'RIMM RGB',
 'ROMM RGB',
 'S-Log',
 'S-Log2',
 'S-Log3',
 'SMPTE 240M',
 'ST 2084',
 'T-Log',
 'V-Log',
 'ViperLog',
 'sRGB']
```

4.5.12 5.5.12 Colour Notation Systems - `colour.notation`

4.5.12.1 5.5.12.1 Munsell Value

```
>>> colour.munsell_value(12.23634268)
4.0824437076525664
>>> sorted(colour.MUNSELL_VALUE_METHODS.keys())
['ASTM D1535',
 'Ladd 1955',
 'McCamy 1987',
 'Moon 1943',
 'Munsell 1933',
 'Priest 1920',
```

(continues on next page)

(continued from previous page)

```
 'Saunderson 1944',
 'astm2008']
```

4.5.12.2 5.5.12.2 Munsell Colour

```
>>> colour.xyY_to_munsell_colour([0.38736945, 0.35751656, 0.59362000])
'4.2YR 8.1/5.3'
>>> colour.munsell_colour_to_xyY('4.2YR 8.1/5.3')
array([ 0.38736945,  0.35751656,  0.59362   ])
```

4.5.13 5.5.13 Optical Phenomena - `colour.phenomena`

```
>>> colour.rayleigh_scattering_sd()
SpectralDistribution([[  3.60000000e+02,   5.99101337e-01],
                      [  3.61000000e+02,   5.92170690e-01],
                      [  3.62000000e+02,   5.85341006e-01],
                      ...
                      [  7.78000000e+02,   2.55208377e-02],
                      [  7.79000000e+02,   2.53887969e-02],
                      [  7.80000000e+02,   2.52576106e-02]],
                     interpolator=Sprague Interpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={'right': None, 'method': 'Constant', 'left': None})
```

4.5.14 5.5.14 Light Quality - `colour.quality`

4.5.14.1 5.5.14.1 Colour Rendering Index

```
>>> colour.colour_quality_scale(colour.ILLUMINANTS_SDS['FL2'])
64.017283509280588
>>> colour.COLOUR_QUALITY_SCALE_METHODS
('NIST CQS 7.4', 'NIST CQS 9.0')
```

4.5.14.2 5.5.14.2 Colour Quality Scale

```
>>> colour.colour_rendering_index(colour.ILLUMINANTS_SDS['FL2'])
64.151520202968015
```

4.5.14.3 5.5.14.3 Academy Spectral Similarity Index (SSI)

```
>>> colour.spectral_similarity_index(colour.ILLUMINANTS_SDS['C'], colour.ILLUMINANTS_SDS['D65'])
94.0
```

4.5.15 5.5.15 Spectral Up-sampling & Reflectance Recovery - colour.recovery

```
>>> colour.XYZ_to_sd([0.20654008, 0.12197225, 0.05136952])
SpectralDistribution([[  3.60000000e+02,   7.73462151e-02],
                      [  3.65000000e+02,   7.73632975e-02],
                      [  3.70000000e+02,   7.74299705e-02],
                      ...
                      [  8.20000000e+02,   3.93126353e-01],
                      [  8.25000000e+02,   3.93158148e-01],
                      [  8.30000000e+02,   3.93163548e-01]],
                     interpolator=SpragueInterpolator,
                     interpolator_args={},
                     extrapolator=Extrapolator,
                     extrapolator_args={'right': None, 'method': 'Constant', 'left': None})

>>> sorted(colour.REFLECTANCE_RECOVERY_METHODS.keys())
['Meng 2015', 'Smits 1999']
```

4.5.16 5.5.16 Correlated Colour Temperature Computation Methods - colour.temperature

```
>>> colour.uv_to_CCT([0.1978, 0.3122])
array([  6.50751282e+03,   3.22335875e-03])
>>> sorted(colour.UV_TO_CCT_METHODS.keys())
['Krystek 1985', 'Ohno 2013', 'Robertson 1968', 'ohno2013', 'robertson1968']
>>> sorted(colour.XY_TO_CCT_METHODS.keys())
['CIE Illuminant D Series', 'Hernandez 1999', 'Kang 2002', 'McCamy 1992', 'daylight', 'hernandez1999
↪', 'kang2002', 'mccamy1992']
```

4.5.17 5.5.17 Colour Volume - colour.volume

```
>>> colour.RGB_colourspace_volume_MonteCarlo(colour.RGB_COLOURSPACE['sRGB'])
821958.30000000005
```

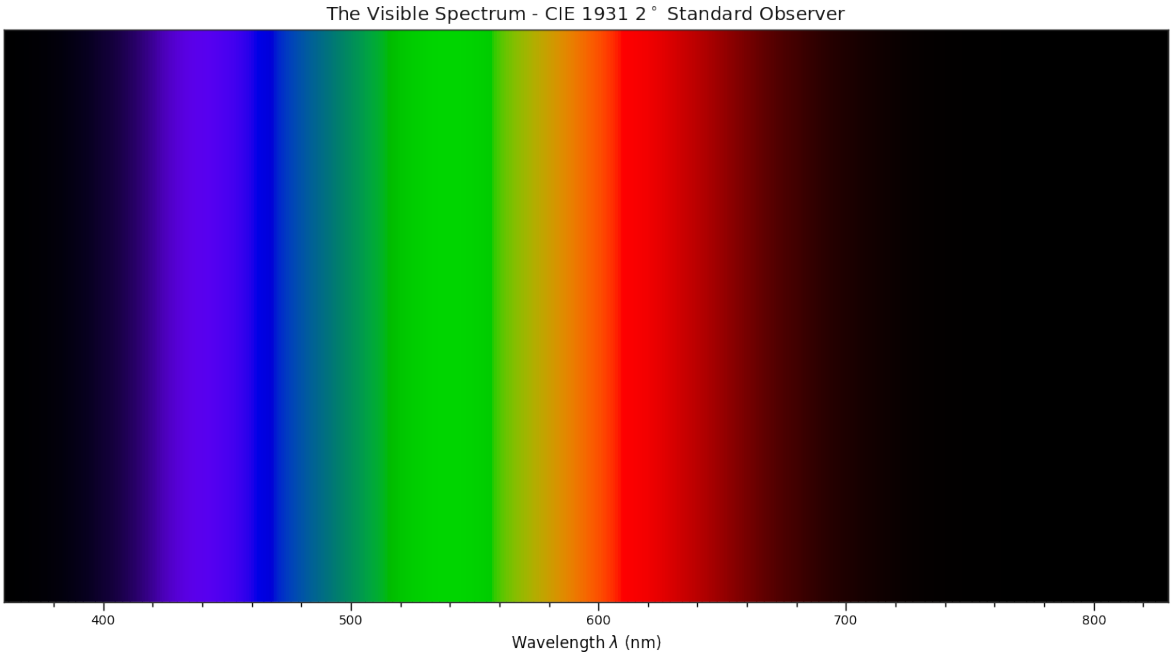
4.5.18 5.5.18 Plotting - colour.plotting

Most of the objects are available from the colour.plotting namespace:

```
>>> from colour.plotting import *
>>> colour_style()
```

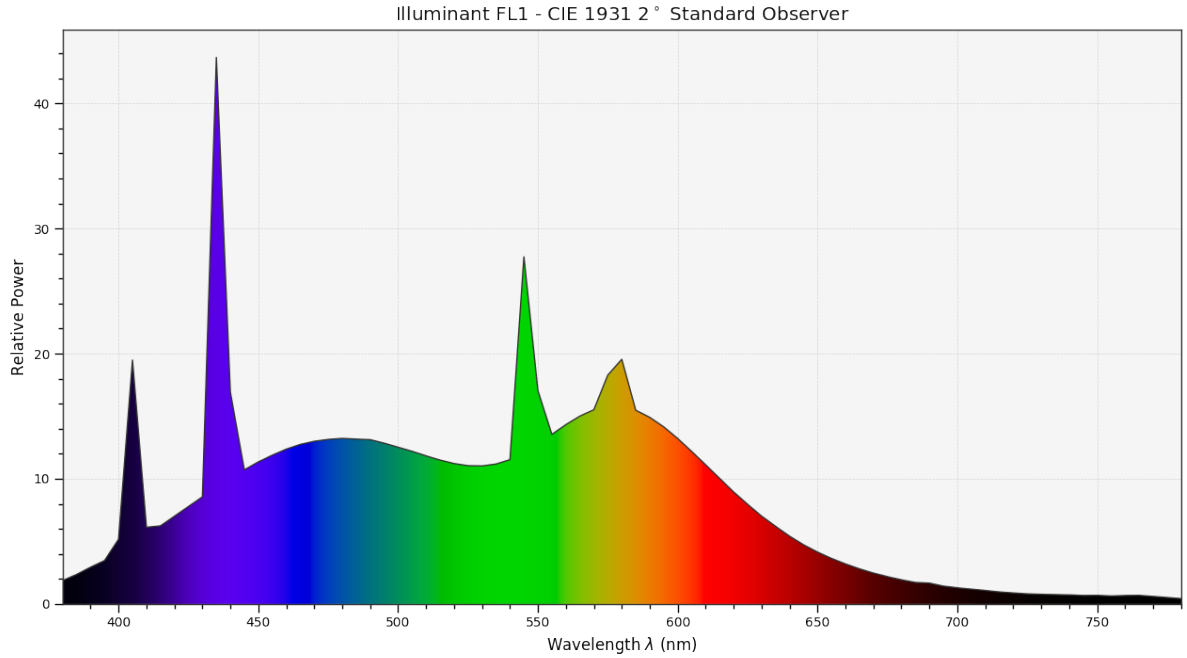
4.5.18.1 5.5.18.1 Visible Spectrum

```
>>> plot_visible_spectrum('CIE 1931 2 Degree Standard Observer')
```



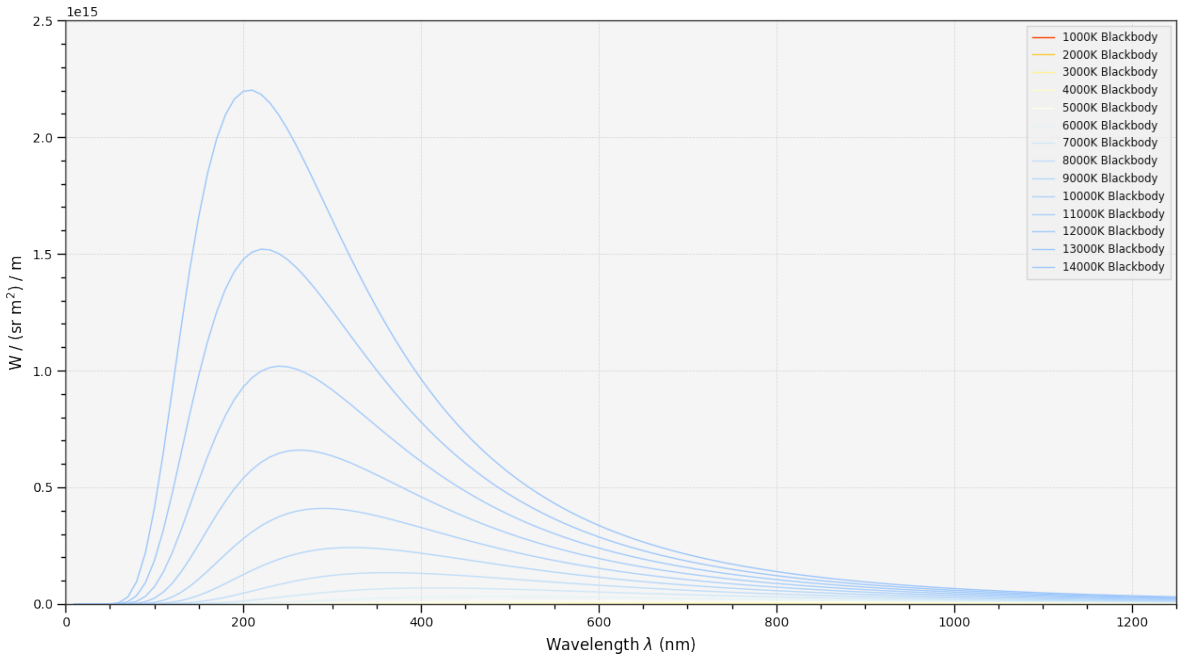
4.5.18.2 5.5.18.2 Spectral Distribution

```
>>> plot_single_illuminant_sd('FL1')
```



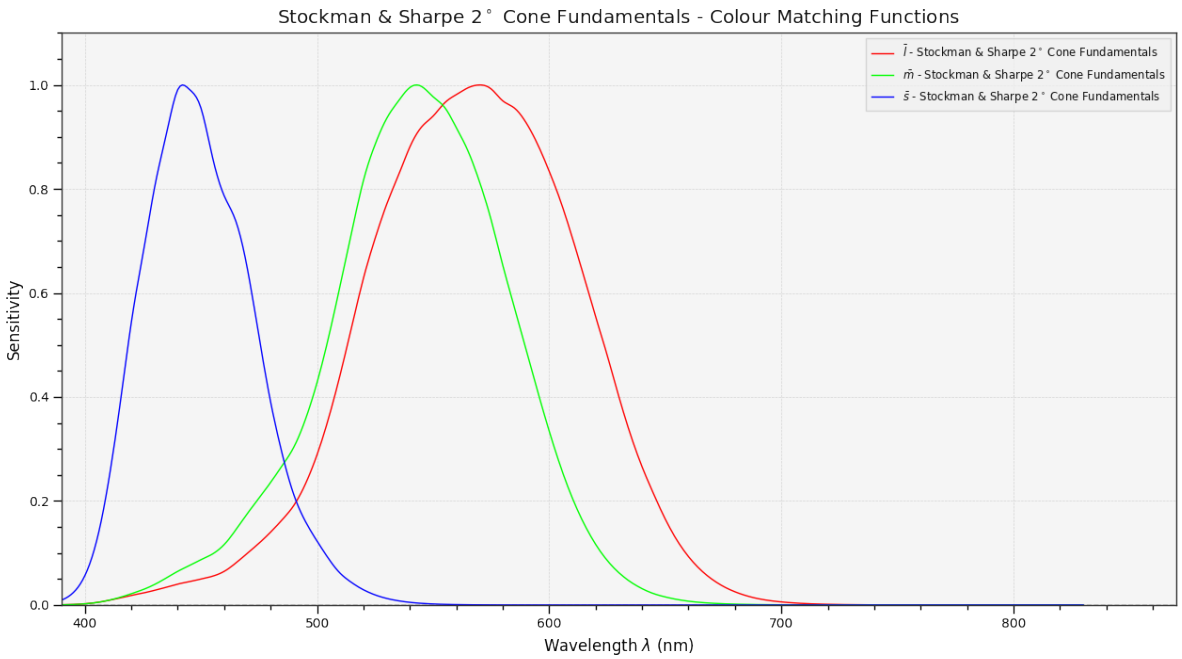
4.5.18.3 5.5.18.3 Blackbody

```
>>> blackbody_sds = [
...     colour.sd_blackbody(i, colour.SpectralShape(0, 10000, 10))
...     for i in range(1000, 15000, 1000)
... ]
>>> plot_multi_sds(
...     blackbody_sds,
...     y_label='W / (sr m$^2$) / m',
...     use_sds_colours=True,
...     normalise_sds_colours=True,
...     legend_location='upper right',
...     bounding_box=(0, 1250, 0, 2.5e15))
```



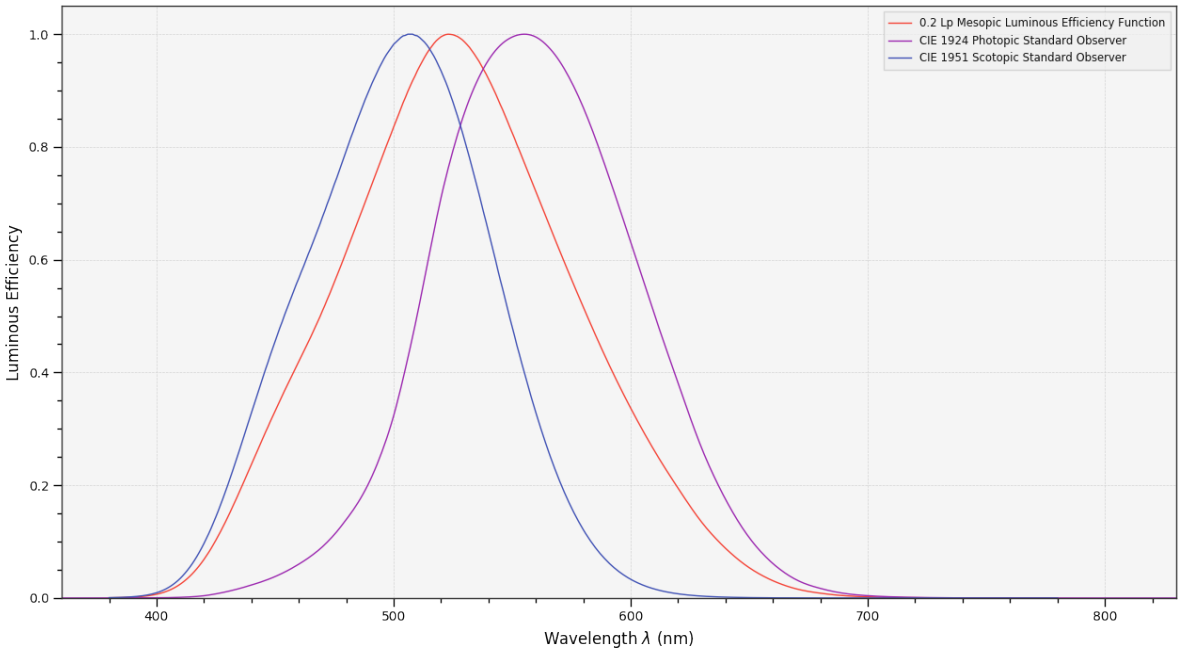
4.5.18.4 5.5.18.4 Colour Matching Functions

```
>>> plot_single_cmfs(
...     'Stockman & Sharpe 2 Degree Cone Fundamentals',
...     y_label='Sensitivity',
...     bounding_box=(390, 870, 0, 1.1))
```


Stockman & Sharpe 2° Cone Fundamentals - Colour Matching Functions
$\bar{l}$ - Stockman & Sharpe 2° Cone Fundamentals
$\bar{m}$ - Stockman & Sharpe 2° Cone Fundamentals
$\bar{s}$ - Stockman & Sharpe 2° Cone Fundamentals
Sensitivity
Wavelength λ (nm)

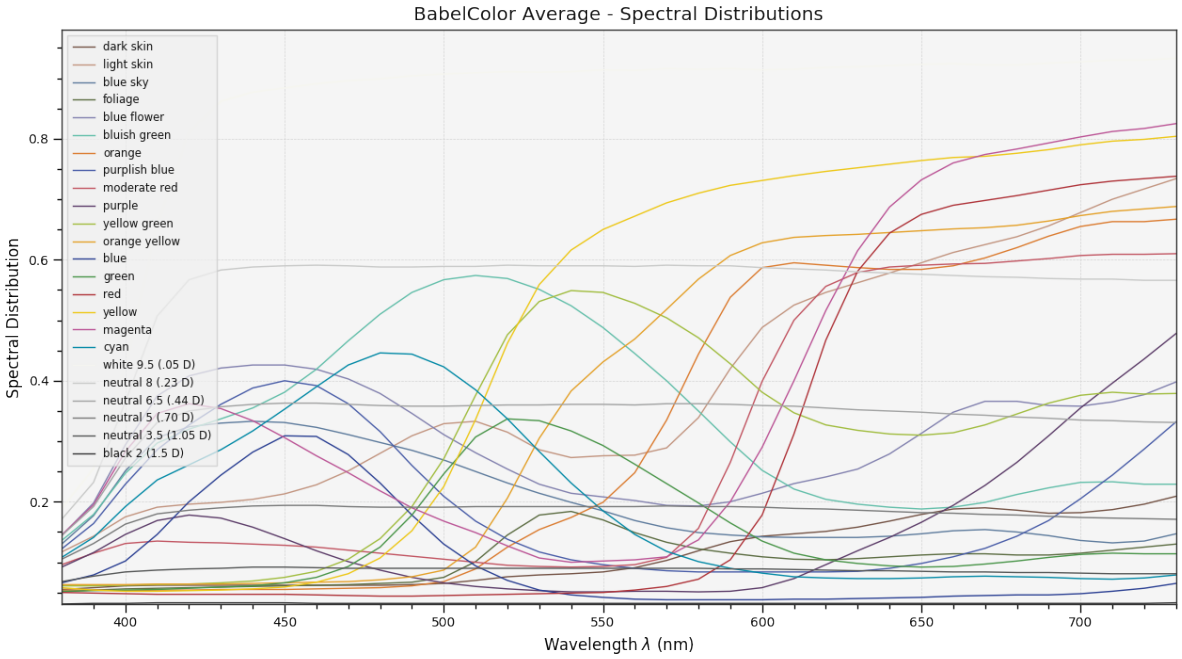
4.5.18.5 5.5.18.5 Luminous Efficiency

```
>>> sd_mesopic_luminous_efficiency_function = (
...     colour.sd_mesopic_luminous_efficiency_function(0.2))
>>> plot_multi_sds(
...     (sd_mesopic_luminous_efficiency_function,
...      colour.PHOTOPIC_LEFS['CIE 1924 Photopic Standard Observer'],
...      colour.SCOTOPIC_LEFS['CIE 1951 Scotopic Standard Observer']),
...     y_label='Luminous Efficiency',
...     legend_location='upper right',
...     y_tighten=True,
...     margins=(0, 0, 0, .1))
```


0.2 Lp Mesopic Luminous Efficiency Function
CIE 1924 Photopic Standard Observer
CIE 1951 Scotopic Standard Observer
Luminous Efficiency
Wavelength λ (nm)

4.5.18.6 5.5.18.6 Colour Checker

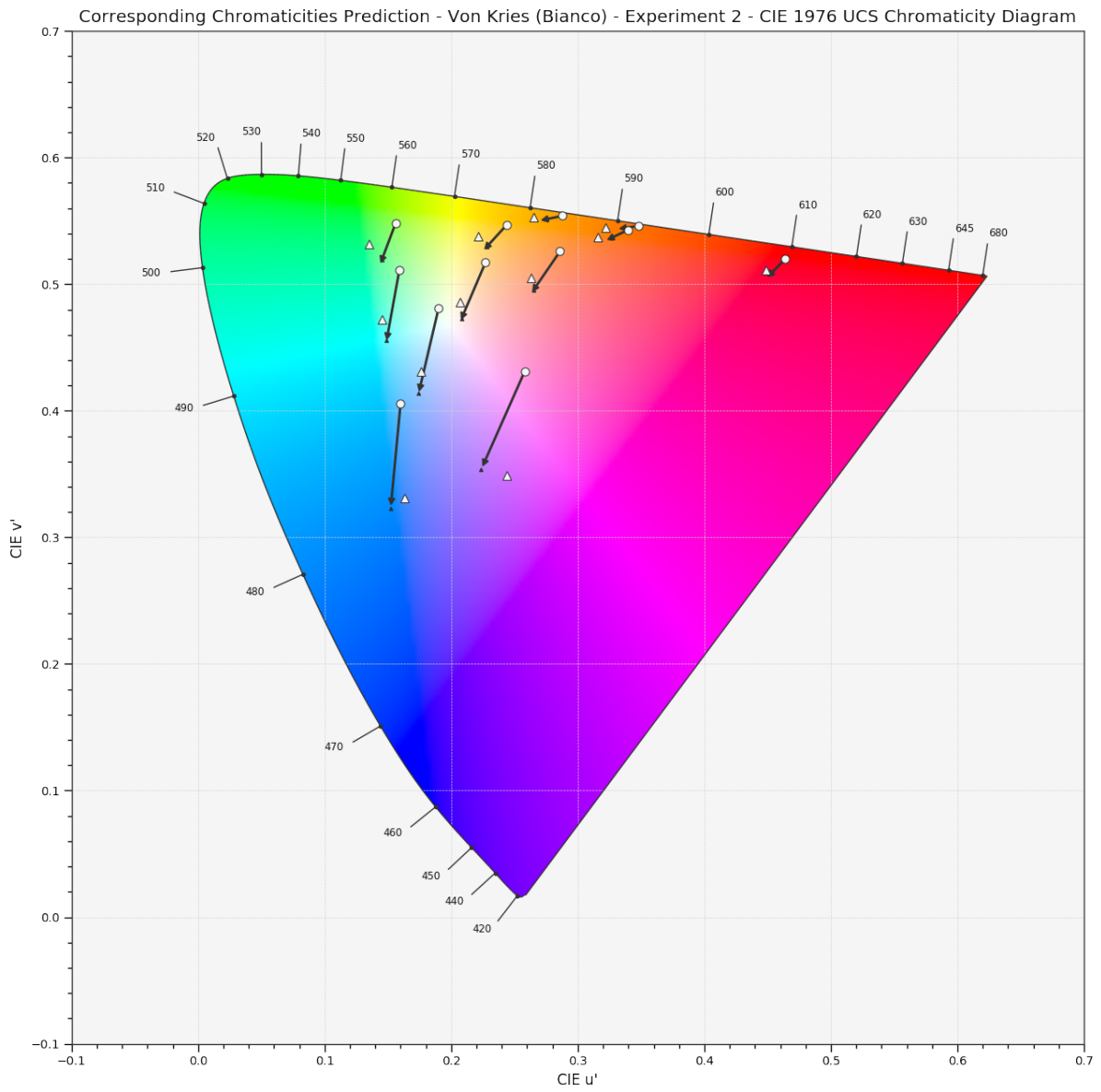
```
>>> from colour.characterisation.dataset.colour_checkers.sds import (
...     COLOURCHECKER_INDEXES_TO_NAMES_MAPPING)
>>> plot_multi_sds(
...     [
...         colour.COLOURCHECKERS_SDS['BabelColor Average'][value]
...         for key, value in sorted(
...             COLOURCHECKER_INDEXES_TO_NAMES_MAPPING.items())
...     ],
...     use_sds_colours=True,
...     title=('BabelColor Average - '
...            'Spectral Distributions'))
```




```
>>> plot_single_colour_checker('ColorChecker 2005', text_parameters={'visible': False})
```

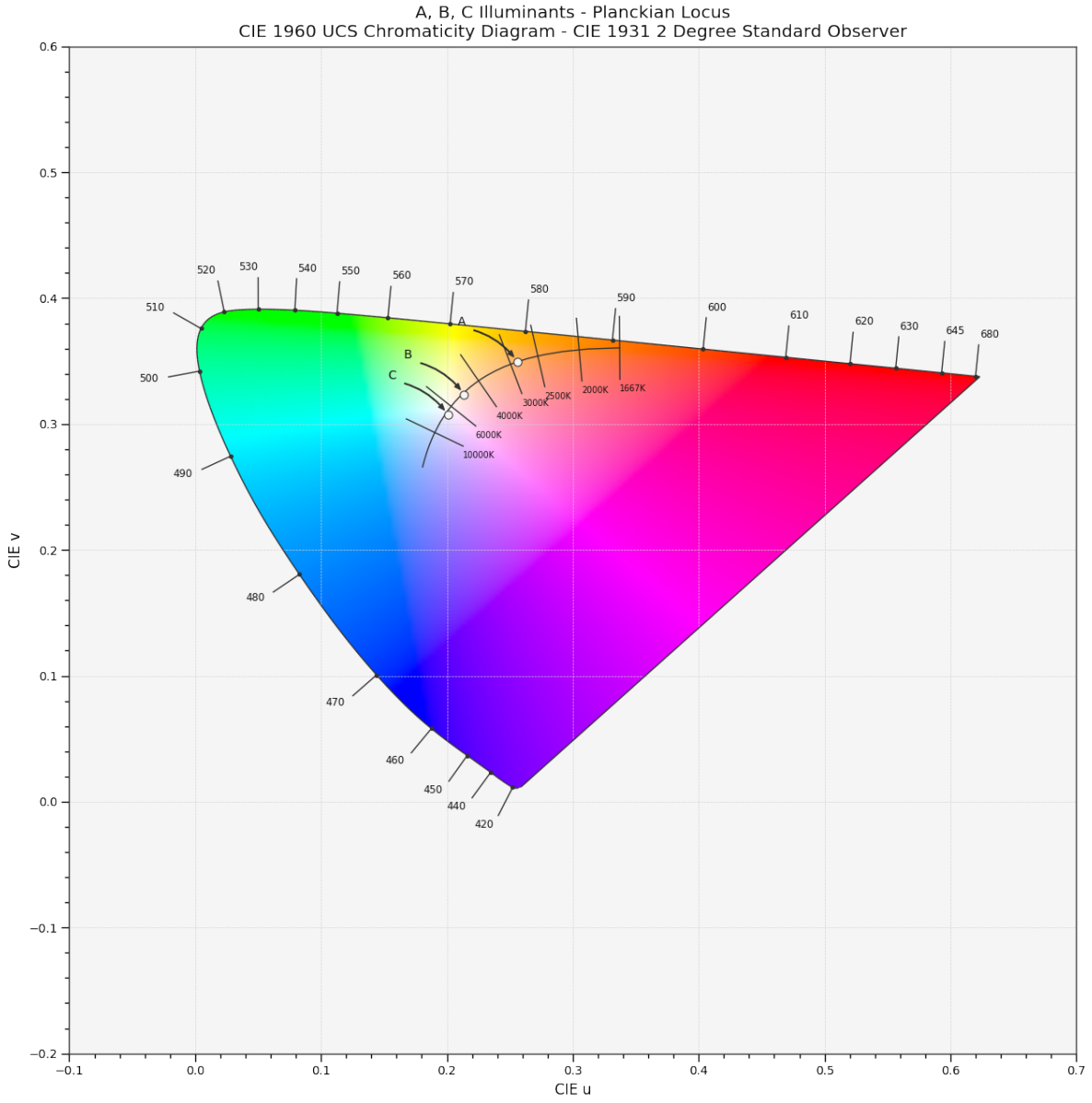
4.5.18.7 5.5.18.7 Chromaticities Prediction

```
>>> plot_corresponding_chromaticities_prediction(2, 'Von Kries', 'Bianco')
```



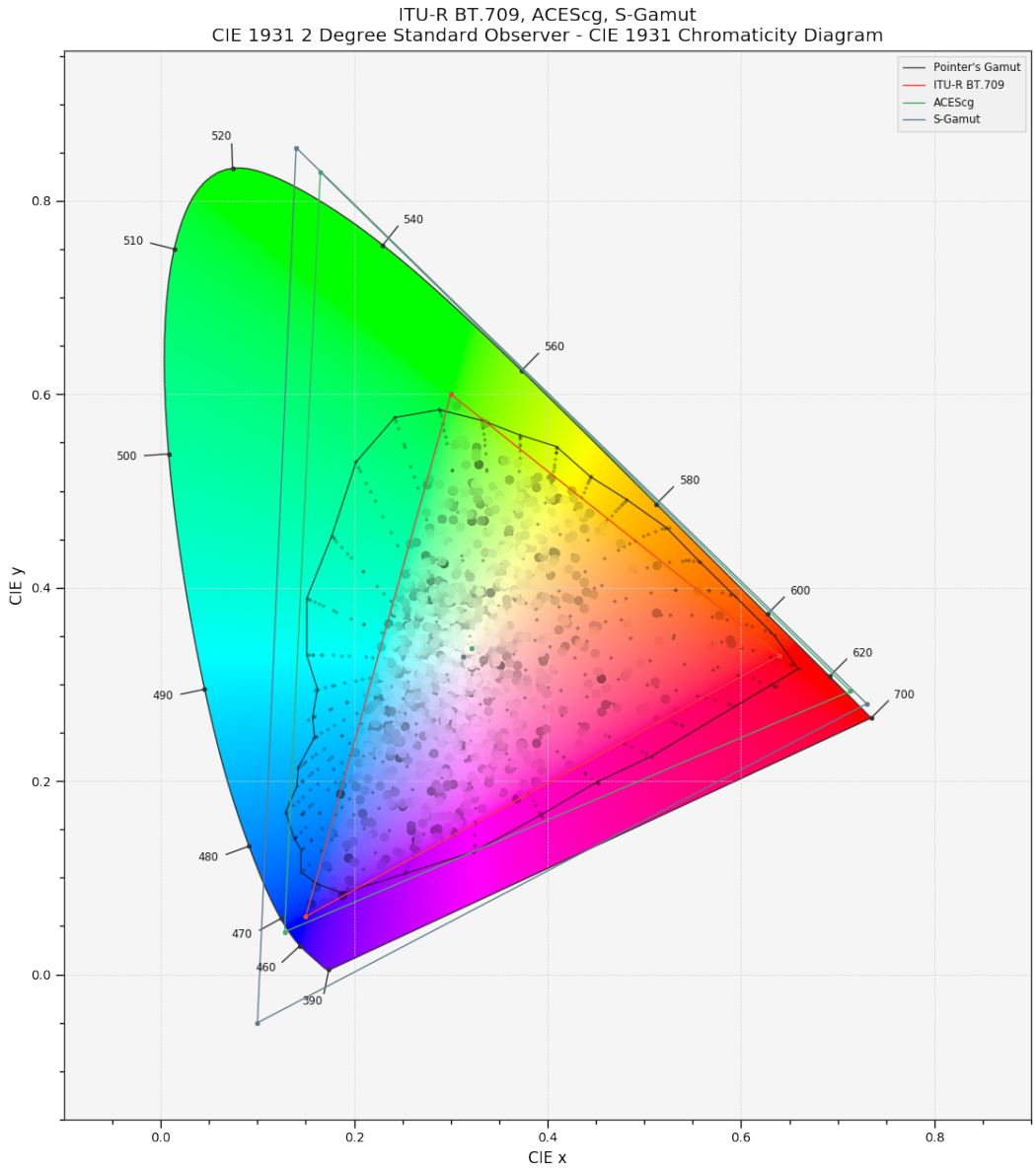
4.5.18.8 5.5.18.8 Colour Temperature

```
>>> plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS(['A', 'B', 'C'])
```



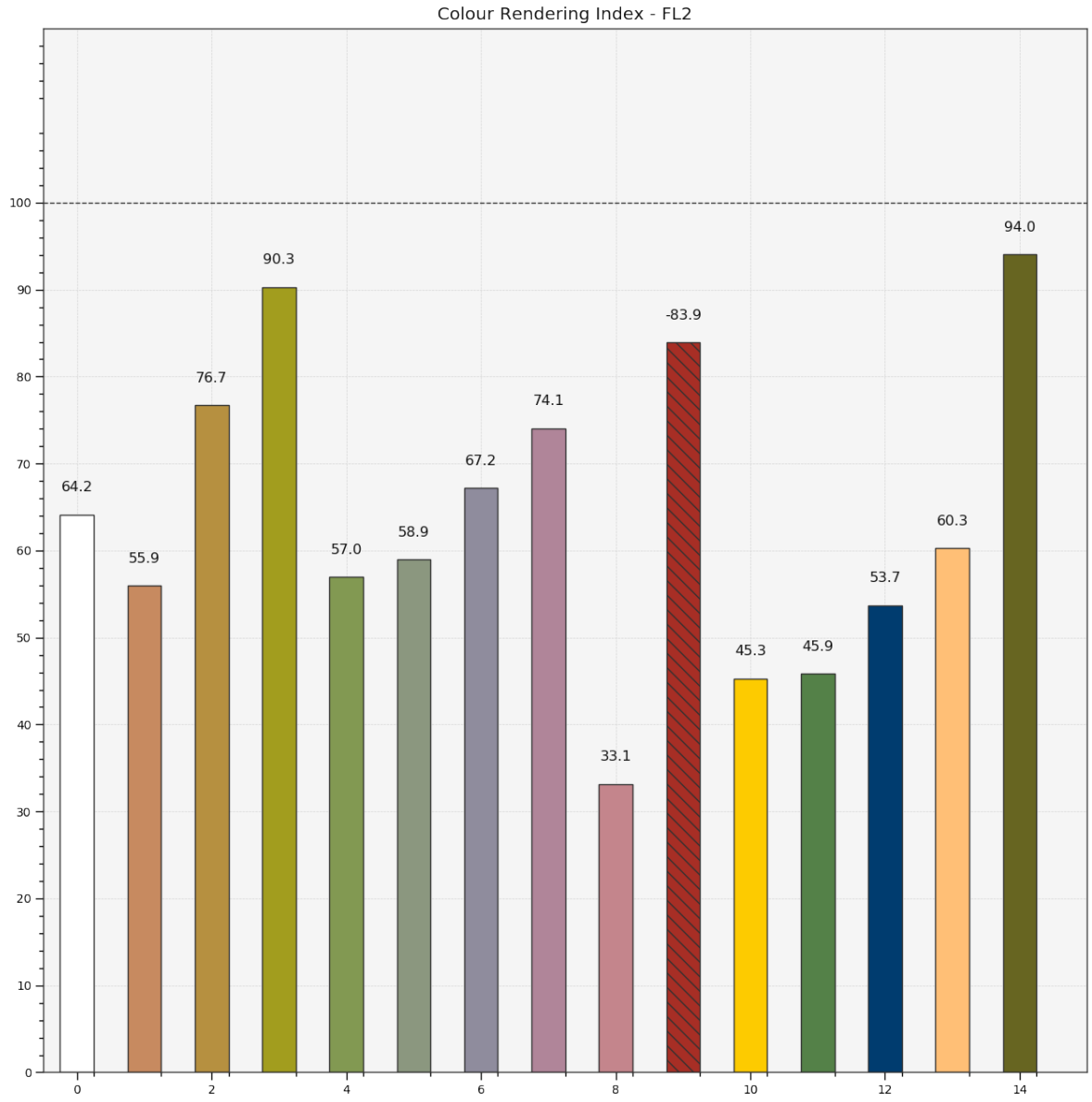
4.5.18.9 5.5.18.9 Chromaticities

```
>>> import numpy as np
>>> RGB = np.random.random((32, 32, 3))
>>> plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931(
...     RGB, 'ITU-R BT.709', colourspaces=['ACEScg', 'S-Gamut'], show_pointer_gamut=True)
```



4.5.18.10 5.5.18.10 Colour Rendering Index

```
>>> plot_single_sd_colour_rendering_index_bars(
...     colour.ILLUMINANTS_SDS['FL2'])
```


Colour Rendering Index - FL2
64.2
55.9
76.7
90.3
57.0
58.9
67.2
74.1
33.1
-83.9
45.3
45.9
53.7
60.3
94.0
0
10
20
30
40
50
60
70
80
90
100
0
2
4
6
8
10
12
14

CHAPTER

FIVE

6 CONTRIBUTING

If you would like to contribute to **Colour**, please refer to the following Contributing guide.

CHAPTER

SIX

7 CHANGES

The changes are viewable on the Releases page.

CHAPTER

SEVEN

8 BIBLIOGRAPHY

The bibliography is available on the Bibliography page.

It is also viewable directly from the repository in BibTeX format.

CHAPTER

EIGHT

9 SEE ALSO

Here is a list of notable colour science packages sorted by languages:

Python

- Colorio by Schlömer, N.
- ColorPy by Kness, M.
- Colorspacious by Smith, N. J., et al.
- python-colormath by Taylor, G., et al.

Go

- go-colorful by Beyer, L., et al.

.NET

- Colourful by Pažourek, T., et al.

Julia

- Colors.jl by Holy, T., et al.

Matlab & Octave

- COLORLAB by Malo, J., et al.
- Psychtoolbox by Brainard, D., et al.
- The Munsell and Kubelka-Munk Toolbox by Centore, P.

CHAPTER

NINE

10 CODE OF CONDUCT

The *Code of Conduct*, adapted from the Contributor Covenant 1.4, is available on the Code of Conduct page.

CHAPTER

TEN

11 ABOUT

Colour by Colour Developers
Copyright © 2013-2020 – Colour Developers – colour-developers@colour-science.org
This software is released under terms of New BSD License:
https://opensource.org/licenses/BSD-3-Clause
https://github.com/colour-science/colour

BIBLIOGRAPHY

[ANS03] ANSI. Specification of ROMM RGB. 2003. URL: http://www.color.org/ROMMRGB.pdf.

[ARR12] ARRI. ALEXA - Log C Curve - Usage in VFX. 2012. URL: https://drive.google.com/open?id=1t73fAG_QpV7hJxoQPYZDWvOojYkYDgvn.

[Bab12a] BabelColor. ColorChecker RGB and spectra. 2012. URL: http://www.babelcolor.com/download/ColorChecker_RGB_and_spectra.xls.

[Bab12b] BabelColor. The ColorChecker (since 1976!). 2012. URL: http://www.babelcolor.com/main_level/ColorChecker.htm.

[Bar99] Peter G. Barten. *Contrast Sensitivity of the Human Eye and Its Effects on Image Quality*. Number 1999. SPIE, December 1999. ISBN 9780819478498. URL: http://ebooks.spiedigitallibrary.org/book.aspx?doi=10.1117/3.353254https://spiedigitallibrary.org/ebooks/PM/Contrast-Sensitivity-of-the-Human-Eye-and-Its-Effects-on/eISBN-9780819478498/10.1117/3.353254, doi:10.1117/3.353254.

[Bar03] Peter G. J. Barten. Formula for the contrast sensitivity of the human eye. In Yoichi Miyake and D. Rene Rasmussen, editors, *Proceedings of SPIE*, volume 5294, 231–238. December 2003. URL: http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?db=pubmed\T1\textbackslash{}&cmd=Retrieve\T1\textbackslash{}&dopt=AbstractPlus\T1\textbackslash{}&list_uids=15798324032973700240related:kPzpPwHsPtsJhttp://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.135956http://proceedings.spiedigitallibrary, doi:10.1117/12.537476.

[BS10] S. Bianco and R. Schettini. Two new von Kries based chromatic adaptation transforms found by numerical optimization. *Color Research & Application*, 35(3):184–192, June 2010. URL: http://doi.wiley.com/10.1002/col.20573, doi:10.1002/col.20573.

[BWDS99] Barry A. Bodhaine, Norman B. Wood, Ellsworth G. Dutton, and James R. Slusser. On Rayleigh Optical Depth Calculations. *Journal of Atmospheric and Oceanic Technology*, 16(11):1854–1861, November 1999. URL: http://journals.ametsoc.org/doi/abs/10.1175/1520-0426%281999%29016%3C1854%3AORODC%3E2.0.CO%3B2, doi:10.1175/1520-0426(1999)016<1854:ORODC>2.0.CO;2.

[Bor17] Tim Borer. Private Discussion with Mansencal, T. and Shaw, N. 2017.

[Boua] Paul Bourke. Intersection point of two line segments in 2 dimensions. URL: http://paulbourke.net/geometry/pointlineplane/.

[Boub] Paul Bourke. Trilinear Interpolation. URL: http://paulbourke.net/miscellaneous/interpolation/.

[Bre87] Edwin J. Breneman. Corresponding chromaticities for different states of adaptation to complex visual fields. *Journal of the Optical Society of America A*, 4(6):1115, June 1987. URL: https://www.osapublishing.org/abstract.cfm?URI=josaa-4-6-1115http://www.opticsinfobase.org/josaa/fulltext.cfm?uri=josaa-4-6-1115\T1\textbackslash{}&id=2783, doi:10.1364/JOSAA.4.001115.

[BS08] Michael H. Brill and Sabine Susstrunk. Repairing gamut problems in CIECAM02: A progress report. *Color Research & Application*, 33(5):424–426, October 2008. URL: http://doi.wiley.com/10.1002/col.20432, doi:10.1002/col.20432.

[Bro09] A. D. Broadbent. Calculation from the Original Experimental Data of the Cie 1931 RGB Standard Observer Spectral Chromaticity Co-Ordinates and Color Matching Functions. 2009. URL: http://www.cis.rit.edu/research/mcsl2/research/broadbent/CIE1931_RGB.pdfhttp://www.cis.rit.edu/mcsl/research/1931.php.

[BB09] Wilhelm Burger and Mark James Burge. *Principles of Digital Image Processing*. Undergraduate Topics in Computer Science. Springer London, London, 2009. ISBN 978-1-84800-194-7. URL: http://link.springer.com/10.1007/978-1-84800-195-4, doi:10.1007/978-1-84800-195-4.

[Can14] Canon. EOS C500 Firmware Update. 2014. URL: https://www.usa.canon.com/internet/portal/us/home/explore/product-showcases/cameras-and-lenses/cinema-eos-firmware/c500.

[Can16] Canon. EOS C300 Mark II - EOS C300 Mark II Input Transform Version 2.0 (for Cinema Gamut / BT.2020). 2016. URL: https://www.usa.canon.com/internet/portal/us/home/support/details/cameras/cinema-eos/eos-c300-mark-ii.

[CTS13] Renbo Cao, H Joel Trussell, and Renzo Shamey. Comparison of the performance of inverse transformation methods from OSA-UCS to CIEXYZ. *Journal of the Optical Society of America A*, 30(8):1508, August 2013. URL: https://www.osapublishing.org/abstract.cfm?URI=josaa-30-8-1508, doi:10.1364/JOSAA.30.001508.

[CSH+18] E.C. Carter, J.D. Schanda, R. Hirschler, S. Jost, M.R. Luo, M. Melgosa, Y. Ohno, M.R. Pointer, D.C. Rich, F. Vienot, L. Whitehead, and J.H. Wold. CIE 015:2018 Colorimetry, 4th Edition. Technical Report, International Commission on Illumination, Vienna, October 2018. URL: http://www.cie.co.at/publications/colorimetry-4th-edition, doi:10.25039/TR.015.2018.

[Cas14] Saullo Castro. Numpy: Fastest way of computing diagonal for each row of a 2d array. 2014. URL: http://stackoverflow.com/questions/26511401/numpy-fastest-way-of-computing-diagonal-for-each-row-of-a-2d-array/26517247#26517247.

[Cen] Paul Centore. The Munsell and Kubelka-Munk Toolbox. URL: http://www.munsellcolourscienceforpainters.com/MunsellAndKubelkaMunkToolbox/MunsellAndKubelkaMunkToolbox.html.

[Cen12] Paul Centore. An open-source inversion algorithm for the Munsell renotation. *Color Research & Application*, 37(6):455–464, December 2012. URL: http://doi.wiley.com/10.1002/col.20715, doi:10.1002/col.20715.

[Cen14a] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - GeneralRoutines/CIELABtoApproxMunsellSpec.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14b] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/ChromDiagHueAngleToMunsellHue.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14c] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/FindHueOnRenotationOvoid.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14d] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MaxChromaForExtrapolatedRenotation.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14e] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToASTMHue.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14f] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellHueToChromDiagHueAngle.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14g] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyForIntegerMunsellValue.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14h] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/MunsellToxyY.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14i] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellRenotationRoutines/xyYtoMunsell.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14j] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/BoundingRenotationHues.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cen14k] Paul Centore. MunsellAndKubelkaMunkToolboxApr2014 - MunsellSystemRoutines/LinearVsRadialInterpOnRenotationOvoid.m. 2014. URL: https://github.com/colour-science/MunsellAndKubelkaMunkToolbox.

[Cha15] Peter Chamberlain. LUT documentation (to create from another program). 2015. URL: https://forum.blackmagicdesign.com/viewtopic.php?f=21\T1\textbackslash{}&t=40284#p232952.

[CWCR04] Vien Cheung, Stephen Westland, David Connah, and Caterina Ripamonti. A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms. *Coloration Technology*, 120(1):19–25, 2004. doi:10.1111/j.1478-4408.2004.tb00201.x.

[CIE] CIE. CIE Spectral Data. URL: http://files.cie.co.at/204.xls.

[CIE04] CIE. CIE 15:2004 Tables Data. 2004. URL: https://law.resource.org/pub/us/cfr/ibr/003/cie.15.2004.tables.xls.

[Cola] Colblindor. Deuteranopia - Red-Green Color Blindness. URL: http://www.color-blindness.com/deuteranopia-red-green-color-blindness/.

[Colb] Colblindor. Protanopia - Red-Green Color Blindness. URL: http://www.color-blindness.com/protanopia-red-green-color-blindness/.

[Colc] Colblindor. Tritanopia - Blue-Yellow Color Blindness. URL: http://www.color-blindness.com/tritanopia-blue-yellow-color-blindness/.

[Cot] Russell Cottrell. The Russell RGB working color space. URL: http://www.russellcottrell.com/photo/downloads/RussellRGB.icc.

[CKM+04] Matthew Cowan, Glenn Kennel, Thomas Maier, Brad Walker, and Matthew Cowan. Constant Sensitivity Experiment to Determine the Bit Depth for Digital Cinema. 2004. URL: http://car.france3.mars.free.fr/FormationINAHD/HDTV/HDTV2007v35/SMPTEnormesetconfs/Contrastm.pdf.

[CVRa] CVRL. CIE (2012) 10-deg XYZ “physiologically-relevant” colour matching functions. URL: http://www.cvrl.org/database/text/cienewxyz/cie2012xyz10.htm.

[CVRb] CVRL. CIE (2012) 2-deg XYZ “physiologically-relevant” colour matching functions. URL: http://www.cvrl.org/database/text/cienewxyz/cie2012xyz2.htm.

[CVRc] CVRL. Luminous efficiency. URL: http://www.cvrl.org/lumindex.htm.

[CVRd] CVRL. New CIE XYZ functions transformed from the CIE (2006) LMS functions. URL: http://cvrl.ioo.ucl.ac.uk/ciexyzpr.htm.

[CVRe] CVRL. Older CIE Standards. URL: http://cvrl.ioo.ucl.ac.uk/cie.htm.

[CVRf] CVRL. Stiles & Burch individual 10-deg colour matching data. URL: http://www.cvrl.org/stilesburch10_ind.htm.

[CVRg] CVRL. Stiles & Burch individual 2-deg colour matching data. URL: http://www.cvrl.org/stilesburch2_ind.htm.

[DFGM15] Maryam Mohammadzadeh Darrodi, Graham Finlayson, Teresa Goodman, and Michal Mackiewicz. Reference data set for camera spectral sensitivity estimation. *Journal of the Optical Society of America A*, 32(3):381, March 2015. URL: https://www.osapublishing.org/abstract.cfm?URI=josaa-32-3-381, doi:10.1364/JOSAA.32.000381.

[DO10] Wendy Davis and Yoshiro Ohno. Color quality scale. *Optical Engineering*, 49(3):033602, March 2010. URL: http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.3360335, doi:10.1117/1.3360335.

[Dji17] Dji. White Paper on D-Log and D-Gamut of DJI Cinema Color System. 2017. URL: https://dl.djicdn.com/downloads/zenmuse+x7/20171010/D-Log_D-Gamut_Whitepaper.pdf.

[Dol16] Dolby. WHAT IS ICTCP? - INTRODUCTION. 2016. URL: https://www.dolby.com/us/en/technologies/dolby-vision/ICtCp-white-paper.pdf.

[Easa] EasyRGB. CMY —> CMYK. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=13#text13.

[Easb] EasyRGB. CMY —> RGB. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=12#text12.

[Easc] EasyRGB. CMYK —> CMY. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=14#text14.

[Easd] EasyRGB. HSL —> RGB. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=19#text19.

[Ease] EasyRGB. HSV —> RGB. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=21#text21.

[Easf] EasyRGB. RGB —> CMY. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=11#text11.

[Easg] EasyRGB. RGB —> HSL. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=18#text18.

[Eash] EasyRGB. RGB —> HSV. URL: http://www.easyrgb.com/index.php?X=MATH\T1\textbackslash{}&H=20#text20.

[EF98] Fritz Ebner and Mark D. Fairchild. Finding constant hue surfaces in color space. In Giordano B. Beretta and Reiner Eschbach, editors, *Proc. SPIE 3300, Color Imaging: Device-Independent Color, Color Hardcopy, and Graphic Arts III, (2 January 1998)*, 107–117. January 1998. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=936964, doi:10.1117/12.298269.

[Erda] U. Murat Erdem. Fast Line Segment Intersection. URL: http://www.mathworks.com/matlabcentral/fileexchange/27205-fast-line-segment-intersection.

[Erdb] Turan Erdogan. How to Calculate Luminosity, Dominant Wavelength, and Excitation Purity. URL: http://www.semrock.com/Data/Sites/1/semrockpdfs/whitepaper_howtocalculateluminositywavelengthandpurity.pdf.

[FW98] M. Fairchild and D. Wyble. Colorimetric Characterization of The Apple Studio Display (flat panel LCD). 1998. URL: http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1922\T1\textbackslash{}&context=article.

[FC11] Mark D Fairchild and Ping-hsu Chen. Brightness, lightness, and specifying color in high-dynamic-range scenes and images. In Susan P. Farnand and Frans Gaykema, editors, *Proc. SPIE 7867, Image Quality and System Performance VIII*, 78670O. January 2011. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.872075, doi:10.1117/12.872075.

[Fai] Mark D. Fairchild. Fairchild YSh. URL: http://rit-mcsl.org/fairchild//files/FairchildYSh.zip.

[Fai91] Mark D. Fairchild. Formulation and testing of an incomplete-chromatic-adaptation model. *Color Research & Application*, 16(4):243–250, August 1991. URL: http://doi.wiley.com/10.1002/col.5080160406, doi:10.1002/col.5080160406.

[Fai96] Mark D. Fairchild. Refinement of the RLAB color space. *Color Research & Application*, 21(5):338–346, October 1996. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199610%2921%3A5%3C338%3A%3AAID-COL3%3E3.0.CO%3B2-Z, doi:10.1002/(SICI)1520-6378(199610)21:5<338::AID-COL3>3.0.CO;2-Z.

[Fai04] Mark D. Fairchild. CIECAM02. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter CIECAM02, pages 289–301. Wiley, 2 edition, 2004.

[Fai13a] Mark D. Fairchild. ATD Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 14.2, pages 5852–5991. Wiley, 3 edition, 2013.

[Fai13b] Mark D. Fairchild. Chromatic Adaptation Models. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4179–4252. Wiley, 3 edition, 2013.

[Fai13c] Mark D. Fairchild. FAIRCHILD'S 1990 MODEL. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4418–4495. Wiley, 3 edition, 2013.

[Fai13d] Mark D. Fairchild. IPT Colourspace. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 20.3, pages 6197–6223. Wiley, 3 edition, 2013.

[Fai13e] Mark D. Fairchild. LLAB Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 14.3, pages 6025–6178. Wiley, 3 edition, 2013.

[Fai13f] Mark D. Fairchild. The Hunt Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 12, pages 5094–5556. Wiley, 3 edition, 2013.

[Fai13g] Mark D. Fairchild. The Nayatani et al. Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 11, pages 4810–5085. Wiley, 3 edition, 2013.

[Fai13h] Mark D. Fairchild. The RLAB Model. In *Color Appearance Models*, The Wiley-IS&T Series in Imaging Science and Technology, chapter 13, pages 5563–5824. Wiley, 3 edition, 2013.

[FW10] Mark D. Fairchild and David R. Wyble. hdr-CIELAB and hdr-IPT: Simple Models for Describing the Color of High-Dynamic-Range and Wide-Color-Gamut Images. In *Proc. of Color and Imaging Conference*, 322–326. 2010. URL: http://www.ingentaconnect.com/content/ist/cic/2010/00002010/00000001/art00057.

[Fai85] Hugh S. Fairman. The calculation of weight factors for tristimulus integration. *Color Research & Application*, 10(4):199–203, 1985. URL: http://doi.wiley.com/10.1002/col.5080100407, doi:10.1002/col.5080100407.

[FBH97] Hugh S. Fairman, Michael H. Brill, and Henry Hemmendinger. How the CIE 1931 color-matching functions were derived from Wright-Guild data. *Color Research & Application*, 22(1):11–23, February 1997. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199702%2922%3A1%3C11%3A%3AAID-COL4%3E3.0.CO%3B2-7, doi:10.1002/(SICI)1520-6378(199702)22:1<11::AID-COL4>3.0.CO;2-7.

[FMH15] Graham D. Finlayson, Michal MacKiewicz, and Anya Hurlbert. Color Correction Using Root-Polynomial Regression. *IEEE Transactions on Image Processing*, 24(5):1460–1470, May 2015. URL: http://ieeexplore.ieee.org/document/7047834/, doi:10.1109/TIP.2015.2405336.

[For18] Alex Forsythe. Private Discussion with Mansencal, T. 2018.

[Fuj16] Fujifilm. F-Log Data Sheet Ver.1.0. 2016. URL: https://www.fujifilm.com/support/digital_cameras/software/lut/pdf/F-Log_DataSheet_E_Ver.1.0.pdf.

[GDY+] Hugo Gaggioni, Patel Dhanendra, Jin Yamashita, N. Kawada, K. Endo, and Curtis Clark. S-Log: A new LUT for digital production mastering and interchange applications. URL: http://pro.sony.com/bbsccms/assets/files/mkt/cinema/solutions/slog_manual.pdf.

[GMRS58] L. G. Glasser, A. H. McKinney, C. D. Reilly, and P. D. Schnelle. Cube-Root Color Coordinate System. *Journal of the Optical Society of America*, 48(10):736, October 1958. URL: https://www.osapublishing.org/abstract.cfm?URI=josa-48-10-736, doi:10.1364/JOSA.48.000736.

[GDM16] GoPro, Haarm-Pieter Duiker, and Thomas Mansencal. Gopro.py. 2016. URL: https://github.com/hpd/OpenColorIO-Configs/blob/master/aces_1.0.3/python/aces_ocio/colorspaces/gopro.py.

[Gut95] S. Lee Guth. Further applications of the ATD model for color vision. In Eric Walowit, editor, *Proc. SPIE 2414, Device-Independent Color Imaging II*, volume 2414, 12–26. April 1995. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=991324, doi:10.1117/12.206546.

[HF98] Radim Halir and Jan Flusser. Numerically Stable Direct Least Squares Fitting Of Ellipses. 1998.

[HernandezAndresLR99] Javier Hernández-Andrés, Raymond L. Lee, and Javier Romero. Calculating correlated color temperatures across the entire gamut of daylight and skylight chromaticities. *Applied Optics*, 38(27):5703, September 1999. URL: https://www.osapublishing.org/abstract.cfm?URI=ao-38-27-5703, doi:10.1364/AO.38.005703.

[Hol] Joseph Holmes. Ekta Space PS 5. URL: https://www.josephholmes.com/userfiles/Ekta_Space_PS5_JHolmes.zip.

[Hou15] Jim Houston. Private Discussion with Mansencal, T. 2015.

[HB95] Po-Chieh Hung and Roy S. Berns. Determination of constant Hue Loci for a CRT gamut and their predictions using color appearance spaces. *Color Research & Application*, 20(5):285–295, October 1995. URL: http://doi.wiley.com/10.1002/col.5080200506, doi:10.1002/col.5080200506.

[Hun04] R.W.G. Hunt. *The Reproduction of Colour*. The Wiley-IS&T Series in Imaging Science and Technology. John Wiley & Sons, Ltd, Chichester, UK, 6 edition, September 2004. ISBN 9780470024270. URL: http://doi.wiley.com/10.1002/0470024275, doi:10.1002/0470024275.

[Hun08a] HunterLab. Hunter L,a,b Color Scale. 2008. URL: http://www.hunterlab.se/wp-content/uploads/2012/11/Hunter-L-a-b.pdf.

[Hun08b] HunterLab. Illuminant Factors in Universal Software and EasyMatch Coatings. 2008. URL: https://support.hunterlab.com/hc/en-us/article_attachments/201437785/an02_02.pdf.

[Hun12] HunterLab. Hunter Rd,a,b Color Scale – History and Application. 2012. URL: https://hunterlabdotcom.files.wordpress.com/2012/07/an-1016-hunter-rd-a-b-color-scale-update-12-07-03.pdf.

[Huta] HutchColor. BestRGB (4 K). URL: http://www.hutchcolor.com/profiles/BestRGB.zip.

[Hutb] HutchColor. DonRGB4 (4 K). URL: http://www.hutchcolor.com/profiles/DonRGB4.zip.

[Hutc] HutchColor. MaxRGB (4 K). URL: http://www.hutchcolor.com/profiles/MaxRGB.zip.

[Hutd] HutchColor. XtremeRGB (4 K). URL: http://www.hutchcolor.com/profiles/XtremeRGB.zip.

[KMH+02] Bongsoon Kang, Ohak Moon, Changhee Hong, Honam Lee, Bonghwan Cho, and Youngsun Kim. Design of advanced color: Temperature control system for HDTV applications. *Journal of the Korean Physical Society*, 41(6):865–871, 2002. URL: http://cat.inist.fr/?aModele=afficheN\T1\textbackslash{}&cpsidt=14448733.

[KPK11] Paul Kienzle, Nikunj Patel, and James Krycka. refl1d.numpyerrors - Refl1D v0.6.19 documentation. 2011. URL: http://www.reflectometry.org/danse/docs/refl1d/_modules/refl1d/numpyerrors.html.

[Kir06] Richard Kirk. Truelight Software Library 2.0. 2006. URL: https://www.filmlight.ltd.uk/pdf/whitepapers/FL-TL-TN-0057-SoftwareLib.pdf.

[Kry85] M Krystek. An algorithm to calculate correlated colour temperature. *Color Research & Application*, 10(1):38–40, 1985. URL: http://doi.wiley.com/10.1002/col.5080100109, doi:10.1002/col.5080100109.

[Lau12] Laurent. Reproducibility of python pseudo-random numbers across systems and versions? 2012. URL: http://stackoverflow.com/questions/8786084/reproducibility-of-python-pseudo-random-numbers-across-systems-and-versions.

[LLW+17] Changjun Li, Zhiqiang Li, Zhifeng Wang, Yang Xu, Ming Ronnier Luo, Guihua Cui, Manuel Melgosa, Michael H Brill, and Michael Pointer. Comprehensive color solutions: CAM16, CAT16, and CAM16-UCS. *Color Research & Application*, 42(6):703–718, December 2017. URL: http://doi.wiley.com/10.1002/col.22131, doi:10.1002/col.22131.

[LLRH02] Changjun Li, Ming Ronnier Luo, Bryan Rigg, and Robert W. G. Hunt. CMC 2000 chromatic adaptation transform: CMCCAT2000. *Color Research & Application*, 27(1):49–58, February 2002. URL: http://doi.wiley.com/10.1002/col.10005, doi:10.1002/col.10005.

[LPLMv07] Changjun Li, Esther Perales, Ming Ronnier Luo, and Francisco Martinez-verdu. The Problem with CAT02 and Its Correction. 2007. URL: https://pdfs.semanticscholar.org/b5a9/0215ad9a1fb6b01f310b3d64305f7c9feb3a.pdf.

[Lin03a] Bruce Lindbloom. Delta E (CIE 1976). 2003. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE76.html.

[Lin03b] Bruce Lindbloom. XYZ to xyY. 2003. URL: http://www.brucelindbloom.com/Eqn_XYZ_to_xyY.html.

[Lin09a] Bruce Lindbloom. Chromatic Adaptation. 2009. URL: http://brucelindbloom.com/Eqn_ChromAdapt.html.

[Lin09b] Bruce Lindbloom. Delta E (CIE 2000). 2009. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE2000.html.

[Lin09c] Bruce Lindbloom. Delta E (CMC). 2009. URL: http://brucelindbloom.com/Eqn_DeltaE_CMC.html.

[Lin09d] Bruce Lindbloom. xyY to XYZ. 2009. URL: http://www.brucelindbloom.com/Eqn_xyY_to_XYZ.html.

[Lin11] Bruce Lindbloom. Delta E (CIE 1994). 2011. URL: http://brucelindbloom.com/Eqn_DeltaE_CIE94.html.

[Lin14] Bruce Lindbloom. RGB Working Space Information. 2014. URL: http://www.brucelindbloom.com/WorkingSpaceInfo.html.

[Lin15] Bruce Lindbloom. About the Lab Gamut. 2015. URL: http://www.brucelindbloom.com/LabGamutDisplayHelp.html.

[LPY+16] Taoran Lu, Fangjun Pu, Peng Yin, Tao Chen, Walt Husak, Jaclyn Pytlarz, Robin Atkins, Jan Froehlich, and Guan-Ming Su. ITP Colour Space and Its Compression Performance for High Dynamic Range and Wide Colour Gamut Video Distribution. *ZTE Communications*, 14(1):32–38, 2016. URL: http://www.cnki.net/kcms/detail/34.1294.TN.20160205.1903.006.html.

[LCL06] M. Ronnier Luo, Guihua Cui, and Changjun Li. Uniform colour spaces based on CIECAM02 colour appearance model. *Color Research & Application*, 31(4):320–330, August 2006. URL: http://doi.wiley.com/10.1002/col.20227, doi:10.1002/col.20227.

[LR99] M. Ronnier Luo and Peter A. Rhodes. Corresponding-colour datasets. *Color Research & Application*, 24(4):295–296, August 1999. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199908%2924%3A4%3C295%3A%3AAID-COL10%3E3.0.CO%3B2-K, doi:10.1002/(SICI)1520-6378(199908)24:4<295::AID-COL10>3.0.CO;2-K.

[LL13] Ming Ronnier Luo and Changjun Li. CIECAM02 and Its Recent Developments. In Christine Fernandez-Maloigne, editor, *Advanced Color Image Processing and Analysis*, pages 19–58. Springer New York, New York, NY, 2013. URL: http://link.springer.com/10.1007/978-1-4419-6190-7, doi:10.1007/978-1-4419-6190-7.

[LLK96] Ming Ronnier Luo, Mei-Chun Lo, and Wen-Guey Kuo. The LLAB (l:c) colour model. *Color Research & Application*, 21(6):412–429, December 1996. URL: http://doi.wiley.com/10.1002/%28SICI%291520-6378%28199612%2921%3A6%3C412%3A%3AAID-COL4%3E3.0.CO%3B2-Z, doi:10.1002/(SICI)1520-6378(199612)21:6<412::AID-COL4>3.0.CO;2-Z.

[LM96] Ming Ronnier Luo and Ján Morovic. Two Unsolved Issues in Colour Management – Colour Appearance and Gamut Mapping. In *Conference: 5th International Conference on High Technology: Imaging Science and Technology – Evolution & Promise*, 136–147. 1996. URL: http://www.researchgate.net/publication/236348295_Two_Unsolved_Issues_in_Colour_Management__Colour_Appearance_and_Gamut_Mapping.

[Mac35] David L. MacAdam. Maximum Visual Efficiency of Colored Materials. *Journal of the Optical Society of America*, 25(11):361–367, November 1935. URL: http://www.opticsinfobase.org/abstract.cfm?URI=josa-25-11-361, doi:10.1364/JOSA.25.000361.

[Mac42] David L. Macadam. Visual Sensitivities to Color Differences in Daylight. *Journal of the Optical Society of America*, 32(5):28, 1942. doi:10.1364/JOSA.32.000247.

[MOF09] G.M. Machado, M.M. Oliveira, and L. Fernandes. A Physiologically-based Model for Simulation of Color Vision Deficiency. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1291–1298, November 2009. URL: http://ieeexplore.ieee.org/document/5290741/, doi:10.1109/TVCG.2009.113.

[Mac10] Gustavo Mello Machado. A model for simulation of color vision deficiency and a color contrast enhancement technique for dichromats. 2010. URL: http://www.lume.ufrgs.br/handle/10183/26950.

[MS03] Henrique Malvar and Gary Sullivan. YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range. 2003. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/Malvar_Sullivan_YCoCg-R_JVT-I014r3-2.pdf.

[Mana] Thomas Mansencal. Lookup. URL: https://github.com/KelSolaar/Foundations/blob/develop/foundations/data_structures.py.

[Manb] Thomas Mansencal. Structure. URL: https://github.com/KelSolaar/Foundations/blob/develop/foundations/data_structures.py.

[Man15] Thomas Mansencal. RED Colourspaces Derivation. 2015. URL: https://www.colour-science.org/posts/red-colourspaces-derivation.

[Man18] Thomas Mansencal. How is the visible gamut bounded? 2018. URL: https://stackoverflow.com/a/48396021/931625.

[Man19] Thomas Mansencal. Colour - Datasets. 2019. URL: https://github.com/colour-science/colour-datasets, doi:10.5281/zenodo.3362520.

[Mel13] Manuel Melgosa. CIE / ISO new standard: CIEDE2000. 2013. URL: http://www.color.org/events/colorimetry/Melgosa_CIEDE2000_Workshop-July4.pdf.

[MSHD15] Johannes Meng, Florian Simon, Johannes Hanika, and Carsten Dachsbacher. Physically Meaningful Rendering using Tristimulus Colours. *Computer Graphics Forum*, 34(4):31–40, July 2015. URL: http://doi.wiley.com/10.1111/cgf.12676, doi:10.1111/cgf.12676.

[Mil14] Scott Miller. A Perceptual EOTF for Extended Dynamic Range Imagery. 2014. URL: https://www.smpte.org/sites/default/files/2014-05-06-EOTF-Miller-1-2-handout.pdf.

[Mor03] Nathan Moroney. A radial sampling of the OSA uniform color scales. *Color and Imaging Conference*, pages 1–14, 2003. URL: http://www.ingentaconnect.com/content/ist/cic/2003/00002003/00000001/art00031.

[MFH+02] Nathan Moroney, Mark D. Fairchild, Robert W. G. Hunt, Changjun Li, Ming Ronnier Luo, and Todd Newman. The CIECAM02 color appearance model. *Color and Imaging Conference*, pages 23–27, 2002. URL: http://www.ingentaconnect.com/content/ist/cic/2002/00002002/00000001/art00006.

[Nat16] Graeme Nattress. Private Discussion with Shaw, N. 2016.

[NSY95] Yoshinobu Nayatani, Hiroaki Sobagaki, and Kenjiro Hashimoto Tadashi Yano. Lightness dependency of chroma scales of a nonlinear color-appearance model and its latest formulation. *Color Research & Application*, 20(3):156–167, June 1995. URL: http://doi.wiley.com/10.1002/col.5080200305, doi:10.1002/col.5080200305.

[NNJ43] Sidney M. Newhall, Dorothy Nickerson, and Deane B. Judd. Final Report of the OSA Subcommittee on the Spacing of the Munsell Colors. *Journal of the Optical Society of America*, 33(7):385, July 1943. URL: https://www.osapublishing.org/abstract.cfm?URI=josa-33-7-385, doi:10.1364/JOSA.33.000385.

[Ohn05] Yoshi Ohno. Spectral design considerations for white LED color rendering. *Optical Engineering*, 44(11):111302, 2005. URL: http://opticalengineering.spiedigitallibrary.org/article.aspx?doi=10.1117/1.2130694, doi:10.1117/1.2130694.

[Ohn14] Yoshiro Ohno. Practical Use and Calculation of CCT and Duv. *LEUKOS*, 10(1):47–55, January 2014. URL: http://www.tandfonline.com/doi/abs/10.1080/15502724.2014.839020, doi:10.1080/15502724.2014.839020.

[OD08] Yoshiro Ohno and Wendy Davis. NIST CQS simulation 7.4. 2008. URL: https://drive.google.com/file/d/1PsuU6QjUJjCX6tQyCud6ul2Tbs8rYWW9/view?usp=sharing.

[OD13] Yoshiro Ohno and Wendy Davis. NIST CQS simulation 9.0. 2013. URL: https://www.researchgate.net/file.PostFileLoader.html?id=5541c498f15bc7cc2c8b4578\T1\textbackslash{}&assetKey=AS%3A273582771376136%401442238623549.

[Oht97] N. Ohta. The basis of color reproduction engineering. 1997.

[Pan14] Panasonic. VARICAM V-Log/V-Gamut. 2014. URL: http://pro-av.panasonic.net/en/varicam/common/pdf/VARICAM_V-Log_V-Gamut.pdf.

[Poi80] Michael R. Pointer. Pointer's Gamut Data. 1980. URL: http://www.cis.rit.edu/research/mcsl2/online/PointerData.xls.

[Rei] Kenneth Reitz. CaseInsensitiveDict. URL: https://github.com/kennethreitz/requests/blob/v1.2.3/requests/structures.py#L37.

[Sae] Saeedn. Extend a line segment a specific distance. URL: http://stackoverflow.com/questions/7740507/extend-a-line-segment-a-specific-distance.

[SCKL17] Muhammad Safdar, Guihua Cui, Youn Jin Kim, and Ming Ronnier Luo. Perceptually uniform color space for image signals including high dynamic range and wide gamut. *Optics Express*, 25(13):15131, June 2017. URL: https://www.osapublishing.org/abstract.cfm?URI=oe-25-13-15131, doi:10.1364/OE.25.015131.

[Sas] Sastanin. How to make scipy.interpolate give an extrapolated result beyond the input range? URL: http://stackoverflow.com/a/2745496/931625.

[SWD05] Gaurav Sharma, Wencheng Wu, and Edul N. Dalal. The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations. *Color Research & Application*, 30(1):21–30, February 2005. URL: http://doi.wiley.com/10.1002/col.20070, doi:10.1002/col.20070.

[SH15] Peter Shirley and David Hart. The prismatic color space for rgb computations. 2015.

[Sir18] Daniele Siragusano. Private Discussion with Shaw, Nick. 2018.

[Smi78] Alvy Ray Smith. Color gamut transform pairs. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques - SIGGRAPH '78*, SIGGRAPH '78, 12–19. New York, New York, USA, 1978. ACM Press. URL: http://portal.acm.org/citation.cfm?doid=800248.807361, doi:10.1145/800248.807361.

[Smi99] Brian Smits. An RGB-to-Spectrum Conversion for Reflectances. *Journal of Graphics Tools*, 4(4):11–22, January 1999. URL: http://www.tandfonline.com/doi/abs/10.1080/10867651.1999.10487511, doi:10.1080/10867651.1999.10487511.

[SWG00] K E Spaulding, G J Woolfe, and E J Giorgianni. Reference Input/Output Medium Metric RGB Color Encodings (RIMM/ROMM RGB). 2000. URL: http://www.photo-lovers.org/pdf/color/romm.pdf.

[Spi15] Nick Spiker. Private Discussion with Mansencal, T. 2015. URL: http://www.invisiblelightimages.com/.

[SS88] E. I. Stearns and R. E. Stearns. An example of a method for correcting radiance data for Bandpass error. *Color Research & Application*, 13(4):257–259, August 1988. URL: http://doi.wiley.com/10.1002/col.5080130410, doi:10.1002/col.5080130410.

[SS00] Andrew Stockman and Lindsay T. Sharpe. Cone Fundamentals. 2000. URL: http://www.cvrl.org/cones.htm.

[SBS99] Sabine Susstrunk, Robert Buckley, and Steve Swen. Standard RGB Color Spaces. 1999.

[SHF00] Sabine E. Susstrunk, Jack M. Holm, and Graham D. Finlayson. Chromatic adaptation performance of different RGB sensors. In Reiner Eschbach and Gabriel G. Marcu, editors, *Photonics West 2001 - Electronic Imaging*, volume 4300, 172–183. December 2000. URL: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=903890, doi:10.1117/12.410788.

[Tho12] Larry Thorpe. CANON-LOG TRANSFER CHARACTERISTIC. 2012. URL: http://downloads.canon.com/CDLC/Canon-Log_Transfer_Characteristic_6-20-2012.pdf.

[Tri15] Tashi Trieu. Private Discussion with Mansencal, T. 2015.

[War16] Greg Ward. Private Discussion with Mansencal, T. 2016.

[WEV02] Greg Ward and Elena Eydelberg-Vileshin. Picture Perfect RGB Rendering Using Spectral Prefiltering and Sharp Color Primaries. *Eurographics workshop on Rendering*, pages 117–124, 2002. URL: http://portal.acm.org/citation.cfm?id=581896.581913%5Cnpapers2://publication/uuid/72F077CB-2366-40E4-901E-5FE35B85BAD6http://dl.acm.org/citation.cfm?id=581913, doi:10.2312/EGWR/EGWR02/117-124.

[WR04] Stephen Westland and Caterina Ripamonti. Table 8.2. In *Computational Colour Science Using MATLAB*, pages 137. John Wiley & Sons, Ltd, Chichester, UK, 1 edition, March 2004. URL: http://doi.wiley.com/10.1002/0470020326, doi:10.1002/0470020326.

[WRC12a] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT2000. In *Computational Colour Science Using MATLAB*, chapter 6.2.3, pages 83–86. 2 edition, 2012.

[WRC12b] Stephen Westland, Caterina Ripamonti, and Vien Cheung. CMCCAT97. In *Computational Colour Science Using MATLAB*, chapter 6.2.2, pages 80. 2 edition, 2012.

[WRC12c] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Correction for Spectral Bandpass. In *Computational Colour Science Using MATLAB*, chapter 4.4, pages 38. 2 edition, 2012.

[WRC12d] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Extrapolation Methods. In *Computational Colour Science Using MATLAB*, chapter 4.4, pages 38. 2 edition, 2012.

[WRC12e] Stephen Westland, Caterina Ripamonti, and Vien Cheung. Interpolation Methods. In *Computational Colour Science Using MATLAB*, chapter 4.3, pages 29–37. 2 edition, 2012.

[Wik] Wikipedia. Ellipse. URL: https://en.wikipedia.org/wiki/Ellipse.

[Wik01a] Wikipedia. Approximation. 2001. URL: http://en.wikipedia.org/wiki/Color_temperature#Approximation.

[Wik01b] Wikipedia. Color temperature. 2001. URL: http://en.wikipedia.org/wiki/Color_temperature.

[Wik01c] Wikipedia. Luminance. 2001. URL: https://en.wikipedia.org/wiki/Luminance.

[Wik01d] Wikipedia. Rayleigh scattering. 2001. URL: http://en.wikipedia.org/wiki/Rayleigh_scattering.

[Wik03a] Wikipedia. HSL and HSV. 2003. URL: http://en.wikipedia.org/wiki/HSL_and_HSV.

[Wik03b] Wikipedia. Lagrange polynomial - Definition. 2003. URL: https://en.wikipedia.org/wiki/Lagrange_polynomial#Definition.

[Wik03c] Wikipedia. Luminosity function. 2003. URL: https://en.wikipedia.org/wiki/Luminosity_function#Details.

[Wik03d] Wikipedia. Mean squared error. 2003. URL: https://en.wikipedia.org/wiki/Mean_squared_error.

[Wik03e] Wikipedia. Michaelis–Menten kinetics. 2003. URL: https://en.wikipedia.org/wiki/Michaelis\T1\textendash{}Menten_kinetics.

[Wik03f] Wikipedia. Vandermonde matrix. 2003. URL: https://en.wikipedia.org/wiki/Vandermonde_matrix.

[Wik04a] Wikipedia. Peak signal-to-noise ratio. 2004. URL: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.

[Wik04b] Wikipedia. Surfaces. 2004. URL: http://en.wikipedia.org/wiki/Gamut#Surfaces.

[Wik04c] Wikipedia. Whiteness. 2004. URL: http://en.wikipedia.org/wiki/Whiteness.

[Wik04d] Wikipedia. Wide-gamut RGB color space. 2004. URL: http://en.wikipedia.org/wiki/Wide-gamut_RGB_color_space.

[Wik04e] Wikipedia. YCbCr. 2004. URL: https://en.wikipedia.org/wiki/YCbCr.

[Wik05a] Wikipedia. CIE 1931 color space. 2005. URL: http://en.wikipedia.org/wiki/CIE_1931_color_space.

[Wik05b] Wikipedia. ISO 31-11. 2005. URL: https://en.wikipedia.org/wiki/ISO_31-11.

[Wik05c] Wikipedia. Lanczos resampling. 2005. URL: https://en.wikipedia.org/wiki/Lanczos_resampling.

[Wik05d] Wikipedia. Luminous Efficacy. 2005. URL: https://en.wikipedia.org/wiki/Luminous_efficacy.

[Wik05e] Wikipedia. Mesopic weighting function. 2005. URL: http://en.wikipedia.org/wiki/Mesopic_vision#Mesopic_weighting_function.

[Wik06a] Wikipedia. List of common coordinate transformations. 2006. URL: http://en.wikipedia.org/wiki/List_of_common_coordinate_transformations.

[Wik06b] Wikipedia. White points of standard illuminants. 2006. URL: http://en.wikipedia.org/wiki/Standard_illuminant#White_points_of_standard_illuminants.

[Wik07a] Wikipedia. CAT02. 2007. URL: http://en.wikipedia.org/wiki/CIECAM02#CAT02.

[Wik07b] Wikipedia. CIECAM02. 2007. URL: http://en.wikipedia.org/wiki/CIECAM02.

[Wik07c] Wikipedia. CIELUV. 2007. URL: http://en.wikipedia.org/wiki/CIELUV.

[Wik07d] Wikipedia. Lightness. 2007. URL: http://en.wikipedia.org/wiki/Lightness.

[Wik07e] Wikipedia. The reverse transformation. 2007. URL: http://en.wikipedia.org/wiki/CIELUV#The_reverse_transformation.

[Wik08a] Wikipedia. CIE 1960 color space. 2008. URL: http://en.wikipedia.org/wiki/CIE_1960_color_space.

[Wik08b] Wikipedia. CIE 1964 color space. 2008. URL: http://en.wikipedia.org/wiki/CIE_1964_color_space.

[Wik08c] Wikipedia. Color difference. 2008. URL: http://en.wikipedia.org/wiki/Color_difference.

[Wik08d] Wikipedia. Relation to CIE XYZ. 2008. URL: http://en.wikipedia.org/wiki/CIE_1960_color_space#Relation_to_CIE_XYZ.

[Wys63] Günter Wyszecki. Proposal for a New Color-Difference Formula. *Journal of the Optical Society of America*, 53(11):1318, November 1963. URL: https://www.osapublishing.org/abstract.cfm?URI=josa-53-11-1318, doi:10.1364/JOSA.53.001318.

[WS00a] Günther Wyszecki and W S Stiles. Equation I(1.2.1). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 8. Wiley, 2000.

[WS00b] Günther Wyszecki and W S Stiles. Table 2(5.4.1) MacAdam Ellipses (Observer PGN) Observed and Calculated on the Basis of a Normal Distribution of Color Matches about a Color Center (Silberstein and MacAdam, 1945). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 309. Wiley, 2000.

[WS00c] Günther Wyszecki and W. S. Stiles. CIE 1976 (L*u*v*)-Space and Color-Difference Formula. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 167. Wiley, 2000.

[WS00d] Günther Wyszecki and W. S. Stiles. CIE Method of Calculating D-Illuminants. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 145–146. Wiley, 2000.

[WS00e] Günther Wyszecki and W. S. Stiles. DISTRIBUTION TEMPERATURE, COLOR TEMPERATURE, AND CORRELATED COLOR TEMPERATURE. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 224–229. Wiley, 2000.

[WS00f] Günther Wyszecki and W. S. Stiles. Integration Replaced by Summation. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 158–163. Wiley, 2000.

[WS00g] Günther Wyszecki and W. S. Stiles. Standard Photometric Observers. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 256–259,395. Wiley, 2000.

[WS00h] Günther Wyszecki and W. S. Stiles. Table 1(3.11) Isotemperature Lines. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 228. Wiley, 2000.

[WS00i] Günther Wyszecki and W. S. Stiles. Table 1(3.3.3). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 138–139. Wiley, 2000.

[WS00j] Günther Wyszecki and W. S. Stiles. Table I(3.7). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 776–777. Wiley, 2000.

[WS00k] Günther Wyszecki and W. S. Stiles. Table I(6.5.3) Whiteness Formulae (Whiteness Measure Denoted by W). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 837–839. Wiley, 2000.

[WS00l] Günther Wyszecki and W. S. Stiles. Table II(3.7). In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 778–779. Wiley, 2000.

[WS00m] Günther Wyszecki and W. S. Stiles. The CIE 1964 Standard Observer. In *Color Science: Concepts and Methods, Quantitative Data and Formulae*, pages 141. Wiley, 2000.

[XR15] X-Rite. New color specifications for ColorChecker SG and Classic Charts. 2015. URL: http://xritephoto.com/ph_product_overview.aspx?ID=938\T1\textbackslash{}&Action=Support\T1\textbackslash{}&SupportID=5884#.

[XRP12] X-Rite and Pantone. Color iQC and Color iMatch Color Calculations Guide. 2012. URL: https://www.xrite.com/-/media/xrite/files/apps_engineering_techdocuments/c/09_color_calculations_en.pdf.

[Yor14] Rory Yorke. Python: Change format of np.array or allow tolerance in in1d function. 2014. URL: http://stackoverflow.com/a/23521245/931625.

[AdobeSystems05] Adobe Systems. Adobe RGB (1998) Color Image Encoding. 2005. URL: http://www.adobe.com/digitalimag/pdfs/AdobeRGB1998.pdf.

[AdobeSystems13a] Adobe Systems. Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::Set_xy_coord. 2013. URL: https://www.adobe.com/support/downloads/dng/dng_sdk.html.

[AdobeSystems13b] Adobe Systems. Adobe DNG Software Development Kit (SDK) - 1.3.0.0 - dng_sdk_1_3/dng_sdk/source/dng_temperature.cpp::dng_temperature::xy_coord. 2013. URL: https://www.adobe.com/support/downloads/dng/dng_sdk.html.

[AdobeSystems13c] Adobe Systems. Cube LUT Specification. 2013. URL: https://drive.google.com/open?id=143Eh08ZYncCAMwJ1q4gWxVOqR_OSWYvs.

[AppleInc19] Apple Inc. displayP3. 2019. URL: https://developer.apple.com/documentation/coregraphics/cgcolorspace/1408916-displayp3.

[AssociationoRIaBusinesses15] Association of Radio Industries and Businesses. Essential Parameter Values for the Extended Image Dynamic Range Television (EIDRTV) System for Programme Production. 2015. URL: https://www.arib.or.jp/english/std_tr/broadcasting/desc/std-b67.html.

[ASTMInternational89] ASTM International. ASTM D1535-89 - Standard Practice for Specifying Color by the Munsell System. 1989. URL: http://www.astm.org/DATABASE.CART/HISTORICAL/D1535-89.htm.

[ASTMInternational07] ASTM International. ASTM D2244-07 - Standard Practice for Calculation of Color Tolerances and Color Differences from Instrumentally Measured Color Coordinates. 2007.

[ASTMInternational08] ASTM International. ASTM D1535-08e1 - Standard Practice for Specifying Color by the Munsell System. 2008. doi:10.1520/D1535-08E01.

[ASTMInternational11] ASTM International. ASTM E2022-11 - Standard Practice for Calculation of Weighting Factors for Tristimulus Integration. 2011. doi:10.1520/E2022-11.

[ASTMInternational15] ASTM International. ASTM E308-15 - Standard Practice for Computing the Colors of Objects by Using the CIE System. 2015. doi:10.1520/E0308-15.

[CIET13294] CIE TC 1-32. CIE 109-1994 A Method of Predicting Corresponding Colours under Different Chromatic and Illuminance Adaptations. 1994. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=34.

[CIET13606] CIE TC 1-36. CIE 170-1:2006 Fundamental Chromaticity Diagram with Physiological Axes - Part 1. 2006. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=48.

[CIET13805a] CIE TC 1-38. 9. INTERPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 9, pages 14–19. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.

[CIET13805b] CIE TC 1-38. 9.2.4 Method of interpolation for uniformly spaced independent variable. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 9.2.4, pages 1–27. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.

[CIET13805c] CIE TC 1-38. EXTRAPOLATION. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter 10, pages 19–20. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.

[CIET13805d] CIE TC 1-38. Table V. Values of the c-coefficients of Equ.s 6 and 7. In *CIE 167:2005 Recommended Practice for Tabulating Spectral Data for Use in Colour Computations*, chapter Table V, pages 19. 2005. URL: http://div1.cie.co.at/?i_ca_id=551\T1\textbackslash{}&pubid=47.

[CIET14804a] CIE TC 1-48. 3.1 Recommendations concerning standard physical data of illuminants. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 3.1, pages 12–13. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804b] CIE TC 1-48. 9.1 Dominant wavelength and purity. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 9.1, pages 32–33. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804c] CIE TC 1-48. APPENDIX E. INFORMATION ON THE USE OF PLANCK'S EQUATION FOR STANDARD AIR. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter APPENDIX E, pages 77–82. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804d] CIE TC 1-48. *CIE 015:2004 Colorimetry, 3rd Edition*. Commission internationale de l'éclairage, 2004. ISBN 978-3-901-90633-6. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804e] CIE TC 1-48. CIE 1976 uniform chromaticity scale diagram (UCS diagram). In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 8.1, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804f] CIE TC 1-48. CIE 1976 uniform colour spaces. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 8.2, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804g] CIE TC 1-48. EXPLANATORY COMMENTS - 5. In *CIE 015:2004 Colorimetry, 3rd Edition*, pages 68–68. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804h] CIE TC 1-48. Extrapolation. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 7.2.2.1, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[CIET14804i] CIE TC 1-48. The evaluation of whiteness. In *CIE 015:2004 Colorimetry, 3rd Edition*, chapter 9.4, pages 24. 2004. URL: http://www.cie.co.at/publications/colorimetry-3rd-edition.

[DigitalCInitiatives07] Digital Cinema Initiatives. Digital Cinema System Specification - Version 1.1. 2007. URL: http://www.dcimovies.com/archives/spec_v1_1/DCI_DCinema_System_Spec_v1_1.pdf.

[EuropeanCInitiative02] European Color Initiative. ECI RGB v2. 2002. URL: http://www.eci.org/_media/downloads/icc_profiles_from_eci/ecirgbv20.zip.

[FiLMiCInc17] FiLMiC Inc. FiLMiC Pro - User Manual v6 - Revision 1. 2017. URL: http://www.filmicpro.com/FilmicProUserManualv6.pdf.

[HewlettPDCompany09] Hewlett-Packard Development Company. Understanding the HP DreamColor LP2480zx DCI-P3 Emulation Color Space. 2009. URL: http://www.hp.com/united-states/campaigns/workstations/pdfs/lp2480zx-dci–p3-emulation.pdf.

[IESCCommitteeTM2714WGroup14] IES Computer Committee and TM-27-14 Working Group. IES Standard Format for the Electronic Transfer of Spectral Data Electronic Transfer of Spectral Data. 2014.

[InternationalECommission99] International Electrotechnical Commission. IEC 61966-2-1:1999 - Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB. 1999. URL: https://webstore.iec.ch/publication/6169.

[InternationalTUnion98] International Telecommunication Union. Recommendation ITU-R BT.470-6 - CONVENTIONAL TELEVISION SYSTEMS. 1998. URL: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.470-6-199811-S!!PDF-E.pdf.

[InternationalTUnion11a] International Telecommunication Union. Recommendation ITU-R BT.1886 - Reference electro-optical transfer function for flat panel displays used in HDTV studio production BT Series Broadcasting service. 2011. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1886-0-201103-I!!PDF-E.pdf.

[InternationalTUnion11b] International Telecommunication Union. Recommendation ITU-R BT.601-7 - Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. 2011. URL: http://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf.

[InternationalTUnion11c] International Telecommunication Union. Recommendation ITU-T T.871 - Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF). 2011. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e\T1\textbackslash{}&id=T-REC-T.871-201105-I!!PDF-E\T1\textbackslash{}&type=items.

[InternationalTUnion15a] International Telecommunication Union. Recommendation ITU-R BT.2020 - Parameter values for ultra-high definition television systems for production and international programme exchange. 2015. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2020-2-201510-I!!PDF-E.pdf.

[InternationalTUnion15b] International Telecommunication Union. Recommendation ITU-R BT.709-6 - Parameter values for the HDTV standards for production and international programme exchange BT Series Broadcasting service. 2015. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.709-6-201506-I!!PDF-E.pdf.

[InternationalTUnion15c] International Telecommunication Union. Report ITU-R BT.2246-4 - The present state of ultra-high definition television BT Series Broadcasting service. 2015.

[InternationalTUnion17] International Telecommunication Union. Recommendation ITU-R BT.2100-1 - Image parameter values for high dynamic range television for use in production and international programme exchange. 2017. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2100-1-201706-I!!PDF-E.pdf.

[InternationalTUnion18] International Telecommunication Union. Recommendation ITU-R BT.2100-2 - Image parameter values for high dynamic range television for use in production and international programme exchange. 2018. URL: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.2100-2-201807-I!!PDF-E.pdf.

[MunsellCSciencea] Munsell Color Science. Macbeth Colorchecker. URL: http://www.rit-mcsl.org/UsefulData/MacbethColorChecker.xls.

[MunsellCScienceb] Munsell Color Science. Munsell Colours Data. URL: http://www.cis.rit.edu/research/mcsl2/online/munsell.php.

[NationalEMAssociation04] National Electrical Manufacturers Association. Digital Imaging and Communications in Medicine (DICOM) Part 14: Grayscale Standard Display Function. 2004. URL: http://medical.nema.org/dicom/2004/04_14PU.PDF.

[RenewableRDCenter03] Renewable Resource Data Center. Reference Solar Spectral Irradiance: ASTM G-173. 2003. URL: http://rredc.nrel.gov/solar/spectra/am1.5/ASTMG173/ASTMG173.html.

[RisingSResearch] Rising Sun Research. cineSpace LUT Library. URL: https://sourceforge.net/projects/cinespacelutlib/.

[SocietyoMPaTEngineers93] Society of Motion Picture and Television Engineers. *RP 177:1993 : Derivation of Basic Television Color Equations*. Volume RP 177:199. The Society of Motion Picture and Television Engineers, January 1993. ISBN 978-1-61482-191-5. URL: http://standards.smpte.org/lookup/doi/10.5594/S9781614821915, doi:10.5594/S9781614821915.

[SocietyoMPaTEngineers99] Society of Motion Picture and Television Engineers. ANSI/SMPTE 240M-1995 - Signal Parameters - 1125-Line High-Definition Production Systems. 1999. URL: http://car.france3.mars.free.fr/HD/INA-26jan06/SMPTEnormesetconfs/s240m.pdf.

[SocietyoMPaTEngineers04] Society of Motion Picture and Television Engineers. *RP 145:2004: SMPTE C Color Monitor Colorimetry*. Volume RP 145:200. The Society of Motion Picture and Television Engineers, January 2004. ISBN 978-1-61482-164-9. URL: http://standards.smpte.org/lookup/doi/10.5594/S9781614821649, doi:10.5594/S9781614821649.

[SocietyoMPaTEngineers14] Society of Motion Picture and Television Engineers. SMPTE ST 2084:2014 - Dynamic Range Electro-Optical Transfer Function of Mastering Reference Displays. 2014. URL: http://www.techstreet.com/products/1883436, doi:10.5594/SMPTE.ST2084.2014.

[SonyCorporationa] Sony Corporation. S-Gamut3_S-Gamut3Cine_Matrix.xlsx. URL: https://community.sony.com/sony/attachments/sony/large-sensor-camera-F5-F55/12359/3/S-Gamut3_S-Gamut3Cine_Matrix.xlsx.

[SonyCorporationb] Sony Corporation. S-Log Whitepaper. URL: http://www.theodoropoulos.info/attachments/076_onS-Log.pdf.

[SonyCorporationc] Sony Corporation. Technical Summary for S-Gamut3.Cine/S-Log3 and S-Gamut3/S-Log3. URL: http://community.sony.com/sony/attachments/sony/large-sensor-camera-F5-F55/12359/2/TechnicalSummary_for_S-Gamut3Cine_S-Gamut3_S-Log3_V1_00.pdf.

[SonyCorporation12] Sony Corporation. S-Log2 Technical Paper. 2012. URL: https://pro.sony.com/bbsccms/assets/files/micro/dmpc/training/S-Log2_Technical_PaperV1_0.pdf.

[SonyImageworks12] Sony Imageworks. Make.py. 2012. URL: https://github.com/imageworks/OpenColorIO-Configs/blob/master/nuke-default/make.py.

[TheAoMPAaSciences19] The Academy of Motion Picture Arts and Sciences. Academy Spectral Similarity Index (SSI): Overview. 2019.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Academy Color Encoding System. URL: http://www.oscars.org/science-technology/council/projects/aces.html.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14a] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2013-001 - ACESproxy, an Integer Log Encoding of ACES Image Data. 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14b] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-003 - ACEScc, A Logarithmic Encoding of ACES Data for use within Color Grading Systems. 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14c] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-004 - Informative Notes on SMPTE ST 2065-1 – Academy Color Encoding Specification (ACES). 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee14d] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Technical Bulletin TB-2014-012 - Academy Color Encoding System Version 1.0 Component Names. 2014. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESPSubcommittee15] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project Subcommittee. Specification S-2014-004 - ACEScg – A Working Space for CGI Render and Compositing. 2015. URL: https://github.com/ampas/aces-dev/tree/master/documents.

[TheAoMPAaSciencesScienceaTCouncilAcademyCESACESProject16] The Academy of Motion Picture Arts and Sciences, Science and Technology Council, and Academy Color Encoding System (ACES) Project. Specification S-2016-001 - ACEScct, A Quasi-Logarithmic Encoding of ACES Data for use within Color Grading Systems. 2016. URL: https://github.com/ampas/aces-dev/tree/v1.0.3/documents.

INDEX

Symbols

__add__() (*colour.continuous.AbstractContinuousFunction method*), 253
__call__() (*colour.KernelInterpolator method*), 53
__call__() (*colour.LinearInterpolator method*), 55
__call__() (*colour.NullInterpolator method*), 57
__call__() (*colour.SpragueInterpolator method*), 58
__class__() (*colour.Extrapolator method*), 51
__contains__() (*colour.SpectralShape method*), 124
__contains__() (*colour.continuous.AbstractContinuousFunction method*), 253
__contains__() (*colour.continuous.MultiSignals method*), 264
__contains__() (*colour.continuous.Signal method*), 257
__contains__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__contains__() (*in module colour*), 506
__delitem__() (*colour.LUTSequence method*), 318
__delitem__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__delitem__() (*in module colour*), 506
__div__() (*colour.continuous.AbstractContinuousFunction method*), 253
__eq__() (*colour.LUTSequence method*), 318
__eq__() (*colour.SpectralShape method*), 125
__eq__() (*colour.continuous.AbstractContinuousFunction method*), 253
__eq__() (*colour.continuous.MultiSignals method*), 264
__eq__() (*colour.continuous.Signal method*), 257
__eq__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__eq__() (*in module colour*), 506
__getitem__() (*colour.LUTSequence method*), 318
__getitem__() (*colour.continuous.AbstractContinuousFunction method*), 253
__getitem__() (*colour.continuous.MultiSignals method*), 264
__getitem__() (*colour.continuous.Signal method*), 257
__getitem__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__getitem__() (*in module colour*), 506
__hash__() (*colour.SpectralShape method*), 124
__hash__() (*colour.continuous.AbstractContinuousFunction method*), 253
__hash__() (*colour.continuous.MultiSignals method*), 264
__hash__() (*colour.continuous.Signal method*), 257
__iadd__() (*colour.continuous.AbstractContinuousFunction method*), 253
__idiv__() (*colour.continuous.AbstractContinuousFunction method*), 253
__imul__() (*colour.continuous.AbstractContinuousFunction method*), 253
__init__() (*colour.ATD95_Specification method*), 81
__init__() (*colour.CAM16_Specification method*), 89
__init__() (*colour.CIECAM02_Specification method*), 85
__init__() (*colour.CorrespondingChromaticitiesPrediction method*), 277
__init__() (*colour.CorrespondingColourDataset method*), 276
__init__() (*colour.Hunt_Specification method*), 93
__init__() (*colour.LLAB_Specification method*), 96
__init__() (*colour.Nayatani95_Specification method*), 99
__init__() (*colour.RLAB_Specification method*), 102
__init__() (*colour.SpectralDistribution method*), 126
__init__() (*colour.adaptation.CMCCAT2000_InductionFactors method*), 46
__init__() (*colour.algebra.LineSegmentsIntersections_Specification method*), 74
__init__() (*colour.algebra.spow_enable method*), 78
__init__() (*colour.appearance.CAM16_InductionFactors method*), 91
__init__() (*colour.appearance.CIECAM02_InductionFactors method*), 86
__init__() (*colour.appearance.LLAB_InductionFactors method*), 97
__init__() (*colour.characterisation.ColourChecker method*), 120
__init__() (*colour.domain_range_scale method*), 665

__init__() (*colour.io.ImageAttribute_Specification method*), 301
__init__() (*colour.quality.CQS_Specification method*), 642
__init__() (*colour.quality.CRI_Specification method*), 640
__init__() (*colour.utilities.disable_multiprocessing method*), 668
__ipow__() (*colour.continuous.AbstractContinuousFunction method*), 253
__isub__() (*colour.continuous.AbstractContinuousFunction method*), 253
__iter__() (*colour.SpectralShape method*), 124
__iter__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__iter__() (*in module colour*), 506
__len__() (*colour.LUTSequence method*), 318
__len__() (*colour.SpectralShape method*), 124
__len__() (*colour.continuous.AbstractContinuousFunction method*), 253
__len__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__len__() (*in module colour*), 506
__mul__() (*colour.continuous.AbstractContinuousFunction method*), 253
__ne__() (*colour.LUTSequence method*), 318
__ne__() (*colour.SpectralShape method*), 125
__ne__() (*colour.continuous.AbstractContinuousFunction method*), 253
__ne__() (*colour.continuous.MultiSignals method*), 264
__ne__() (*colour.continuous.Signal method*), 257
__ne__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__ne__() (*in module colour*), 506
__pow__() (*colour.continuous.AbstractContinuousFunction method*), 253
__repr__() (*colour.LUTSequence method*), 318
__repr__() (*colour.RGB_Colourspace method*), 397
__repr__() (*colour.SpectralShape method*), 124
__repr__() (*colour.continuous.AbstractContinuousFunction method*), 253
__repr__() (*colour.continuous.MultiSignals method*), 264
__repr__() (*colour.continuous.Signal method*), 257
__repr__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__repr__() (*in module colour*), 506
__repr__() (*in module colour.models*), 415
__setitem__() (*colour.LUTSequence method*), 318
__setitem__() (*colour.continuous.AbstractContinuousFunction method*), 253
__setitem__() (*colour.continuous.MultiSignals method*), 264
__setitem__() (*colour.continuous.Signal method*), 257
__setitem__() (*colour.utilities.CaseInsensitiveMapping method*), 697
__setitem__() (*in module colour*), 506
__str__() (*colour.LUTSequence method*), 318
__str__() (*colour.RGB_Colourspace method*), 397
__str__() (*colour.SpectralShape method*), 124
__str__() (*colour.continuous.AbstractContinuousFunction method*), 253
__str__() (*colour.continuous.MultiSignals method*), 264
__str__() (*colour.continuous.Signal method*), 257
__str__() (*in module colour.models*), 415
__sub__() (*colour.continuous.AbstractContinuousFunction method*), 253

A

absolute_tolerance (*colour.NullInterpolator attribute*), 57
absolute_tolerance() (*colour.NullInterpolator property*), 57
AbstractContinuousFunction (*class in colour.continuous*), 253
AbstractLUTSequenceOperator (*class in colour.io*), 322
ACES_2065_1_COLOURSPACE (*in module colour.models*), 402
ACES_CC_COLOURSPACE (*in module colour.models*), 403
ACES_CCT_COLOURSPACE (*in module colour.models*), 403
ACES_CG_COLOURSPACE (*in module colour.models*), 404
ACES_PROXY_COLOURSPACE (*in module colour.models*), 403
ACES_RICD (*in module colour.models*), 501
adjust_tristimulus_weighting_factors_ASTME308() (*in module colour.colorimetry*), 185
ADOBE_RGB_1998_COLOURSPACE (*in module colour.models*), 404
ADOBE_WIDE_GAMUT_RGB_COLOURSPACE (*in module colour.models*), 404
ALEXA_WIDE_GAMUT_COLOURSPACE (*in module colour.models*), 404
align() (*colour.MultiSpectralDistributions method*), 139
align() (*colour.SpectralDistribution method*), 126, 127
anomalous_trichromacy_cmfs_Machado2009() (*in module colour*), 104
anomalous_trichromacy_matrix_Machado2009() (*in module colour*), 105
APPLE_RGB_COLOURSPACE (*in module colour.models*), 405
apply() (*colour.io.AbstractLUTSequenceOperator method*), 322
apply() (*colour.LUT1D method*), 305, 306
apply() (*colour.LUT3D method*), 313
apply() (*colour.LUT3x1D method*), 308, 309
apply() (*colour.LUTSequence method*), 318, 319
arithmetical_operation()

(*colour.continuous.AbstractContinuousFunction method*), 253, 254
arithmetical_operation() (*colour.continuous.MultiSignals method*), 264, 267
arithmetical_operation() (*colour.continuous.Signal method*), 257, 258
artist() (*in module colour.plotting*), 544
as_array() (*in module colour.utilities*), 682
as_float() (*in module colour.utilities*), 684
as_float_array() (*in module colour.utilities*), 683
as_int() (*in module colour.utilities*), 684
as_int_array() (*in module colour.utilities*), 683
as_LUT() (*colour.LUT1D method*), 305, 306
as_LUT() (*colour.LUT3D method*), 313, 314
as_LUT() (*colour.LUT3x1D method*), 308, 310
as_namedtuple() (*in module colour.utilities*), 685
as_numeric() (*in module colour.utilities*), 683
ASTME308_PRACTISE_SHAPE (*in module colour*), 150
ATD95_Specification (*class in colour*), 81
augmented_matrix_Cheung2004() (*in module colour.characterisation*), 113
AVOGADRO_CONSTANT (*in module colour.constants*), 243

B

bandpass_correction() (*in module colour*), 193
BANDPASS_CORRECTION_METHODS (*in module colour*), 194
bandpass_correction_Stearns1988() (*in module colour.colorimetry*), 194
bandwidth_corrected (*colour.SpectralDistribution_IESTM2714 attribute*), 335
bandwidth_corrected() (*colour.SpectralDistribution_IESTM2714 property*), 335
bandwidth_FWHM (*colour.SpectralDistribution_IESTM2714 attribute*), 335
bandwidth_FWHM() (*colour.SpectralDistribution_IESTM2714 property*), 335
batch() (*in module colour.utilities*), 668
BEST_RGB_COLOURSPACE (*in module colour.models*), 405
BETA_RGB_COLOURSPACE (*in module colour.models*), 405
blackbody_spectral_radiance() (*in module colour.colorimetry*), 167
BOLTZMANN_CONSTANT (*in module colour.constants*), 243
boundaries (*colour.SpectralShape attribute*), 124
boundaries() (*colour.SpectralShape property*), 125
BRADFORD_CAT (*in module colour.adaptation*), 48
BRENEMAN_EXPERIMENTS (*in module colour*), 278
BRENEMAN_EXPERIMENTS_PRIMARIES_CHROMATICITIES (*in module colour*), 278
BS_CAT (*in module colour.adaptation*), 48
BS_PC_CAT (*in module colour.adaptation*), 48
BT2020_COLOURSPACE (*in module colour.models*), 406
BT2100_HLG_EOTF_INVERSE_METHODS (*in module colour.models*), 447
BT2100_HLG_EOTF_METHODS (*in module colour.models*), 446
BT2100_HLG_OOTF_INVERSE_METHODS (*in module colour.models*), 457
BT2100_HLG_OOTF_METHODS (*in module colour.models*), 456
BT470_525_COLOURSPACE (*in module colour.models*), 405
BT470_625_COLOURSPACE (*in module colour.models*), 406
BT709_COLOURSPACE (*in module colour.models*), 406

C

CAM02LCD_to_JMh_CIECAM02() (*in module colour*), 367
CAM02SCD_to_JMh_CIECAM02() (*in module colour*), 369
CAM02UCS_to_JMh_CIECAM02() (*in module colour*), 371
CAM16_InductionFactors (*class in colour.appearance*), 90
CAM16_Specification (*class in colour*), 89
CAM16_to_XYZ() (*in module colour*), 88
CAM16_VIEWING_CONDITIONS (*in module colour*), 90
CAM16LCD_to_JMh_CAM16() (*in module colour*), 372
CAM16SCD_to_JMh_CAM16() (*in module colour*), 374
CAM16UCS_to_JMh_CAM16() (*in module colour*), 375
camera() (*in module colour.plotting*), 545
CAMERAS_RGB_SPECTRAL_SENSITIVITIES (*in module colour*), 122
cartesian_to_cylindrical() (*in module colour.algebra*), 68
cartesian_to_polar() (*in module colour.algebra*), 67
cartesian_to_spherical() (*in module colour.algebra*), 66
CaseInsensitiveMapping (*class in colour.utilities*), 697
CAT02_BRILL_CAT (*in module colour.adaptation*), 48
CAT02_CAT (*in module colour.adaptation*), 49
CCT_to_uv() (*in module colour*), 652
CCT_to_uv_Krystek1985() (*in module colour.temperature*), 657
CCT_TO_UV_METHODS (*in module colour*), 652
CCT_to_uv_Ohno2013() (*in module colour.temperature*), 659
CCT_to_uv_Robertson1968() (*in module colour.temperature*), 656
CCT_to_xy() (*in module colour*), 654
CCT_to_xy_CIE_D() (*in module colour.temperature*), 664
CCT_to_xy_Hernandez1999() (*in module colour.temperature*), 661

CCT_to_xy_Kang2002() (*in module colour.temperature*), 663
CCT_to_xy_McCamy1992() (*in module colour.temperature*), 660
CCT_TO_XY_METHODS (*in module colour*), 654
cctf_decoding (*colour.RGB_Colourspace attribute*), 397
cctf_decoding (*in module colour.models*), 415
cctf_decoding() (*colour.RGB_Colourspace property*), 398
cctf_decoding() (*in module colour*), 419
cctf_decoding_ProPhotoRGB() (*in module colour.models*), 425
cctf_decoding_RIMMRGB() (*in module colour.models*), 423
cctf_decoding_ROMMRGB() (*in module colour.models*), 421
CCTF_DECODINGS (*in module colour*), 420
cctf_encoding (*colour.RGB_Colourspace attribute*), 397
cctf_encoding (*in module colour.models*), 415
cctf_encoding() (*colour.RGB_Colourspace property*), 398
cctf_encoding() (*in module colour*), 418
cctf_encoding_ProPhotoRGB() (*in module colour.models*), 424
cctf_encoding_RIMMRGB() (*in module colour.models*), 422
cctf_encoding_ROMMRGB() (*in module colour.models*), 421
CCTF_ENCODINGS (*in module colour*), 419
centroid() (*in module colour.utilities*), 693
chromatic_adaptation() (*in module colour*), 36
chromatic_adaptation_CIE1994() (*in module colour.adaptation*), 41
chromatic_adaptation_CMCCAT2000() (*in module colour.adaptation*), 42
chromatic_adaptation_Fairchild1990() (*in module colour.adaptation*), 40
chromatic_adaptation_forward_CMCCAT2000() (*in module colour.adaptation*), 44
chromatic_adaptation_inverse_CMCCAT2000() (*in module colour.adaptation*), 45
chromatic_adaptation_matrix_VonKries() (*in module colour.adaptation*), 49
CHROMATIC_ADAPTATION_METHODS (*in module colour*), 38
CHROMATIC_ADAPTATION_TRANSFORMS (*in module colour*), 39
CHROMATIC_ADAPTATION_TRANSFORMS (*in module colour.adaptation*), 48
chromatic_adaptation_VonKries() (*in module colour.adaptation*), 47
chromatically_adapt() (*colour.RGB_Colourspace method*), 397, 398
chromatically_adapt() (*in module colour.models*), 415
chromatically_adapted_primaries() (*in module colour*), 393
CIE_RGB_COLOURSPACE (*in module colour.models*), 406
CIECAM02_InductionFactors (*class in colour.appearance*), 86
CIECAM02_Specification (*class in colour*), 85
CIECAM02_to_XYZ() (*in module colour*), 83
CIECAM02_VIEWING_CONDITIONS (*in module colour*), 86
CINEMA_GAMUT_COLOURSPACE (*in module colour.models*), 407
closest() (*in module colour.utilities*), 686
closest_indexes() (*in module colour.utilities*), 685
CMCCAT2000_CAT (*in module colour.adaptation*), 49
CMCCAT2000_InductionFactors (*class in colour.adaptation*), 46
CMCCAT2000_VIEWING_CONDITIONS (*in module colour*), 39
CMCCAT2000_VIEWING_CONDITIONS (*in module colour.adaptation*), 43
CMCCAT97_CAT (*in module colour.adaptation*), 49
CMFS (*in module colour*), 197
CMY_to_CMYK() (*in module colour*), 520
CMY_to_RGB() (*in module colour*), 519
CMYK_to_CMY() (*in module colour*), 520
COLOR_MATCH_RGB_COLOURSPACE (*in module colour.models*), 407
colorimetric_purity() (*in module colour*), 207
colour_correction() (*in module colour*), 111
colour_correction_Cheung2004() (*in module colour.characterisation*), 117
colour_correction_Finlayson2015() (*in module colour.characterisation*), 118
colour_correction_matrix() (*in module colour*), 109
colour_correction_matrix_Cheung2004() (*in module colour.characterisation*), 115
colour_correction_matrix_Finlayson2015() (*in module colour.characterisation*), 116
COLOUR_CORRECTION_MATRIX_METHODS (*in module colour*), 109
colour_correction_matrix_Vandermonde() (*in module colour.characterisation*), 116
COLOUR_CORRECTION_METHODS (*in module colour*), 111
colour_correction_Vandermonde() (*in module colour.characterisation*), 118
colour_cycle() (*in module colour.plotting*), 544
colour_quality_scale() (*in module colour*), 641
COLOUR_QUALITY_SCALE_METHODS (*in module colour*), 641
colour_rendering_index() (*in module colour*), 639
colour_style() (*in module colour.plotting*), 544
ColourChecker (*class in colour.characterisation*), 120
COLOURCHECKERS (*in module colour*), 119
COLOURCHECKERS_SDS (*in module colour*), 120

ColourRuntimeWarning (*class in colour.utilities*), 704
ColourUsageWarning (*class in colour.utilities*), 704
ColourWarning (*class in colour.utilities*), 704
common_colourspace_model_axis_reorder() (*in module colour.plotting.models*), 605
complementary_wavelength() (*in module colour*), 205
contrast_sensitivity_function() (*in module colour*), 245
contrast_sensitivity_function_Barten1999() (*in module colour.contrast*), 246
CONTRAST_SENSITIVITY_METHODS (*in module colour*), 246
convert() (*in module colour*), 295
convert_bit_depth() (*in module colour.io*), 302
copy() (*colour.continuous.AbstractContinuousFunction method*), 254
copy() (*colour.LUTSequence method*), 318, 319
copy() (*colour.RGB_Colourspace method*), 397, 399
copy() (*colour.utilities.CaseInsensitiveMapping method*), 697
copy() (*in module colour*), 506
copy() (*in module colour.models*), 415
corresponding_chromaticities_prediction() (*in module colour*), 275
corresponding_chromaticities_prediction_CIE1994() (*in module colour.corresponding*), 279
corresponding_chromaticities_prediction_CMCCAT2000() (*in module colour.corresponding*), 280
corresponding_chromaticities_prediction_Fairchild1990() (*in module colour.corresponding*), 279
CORRESPONDING_CHROMATICITIES_PREDICTION_MODELS (*in module colour*), 276
corresponding_chromaticities_prediction_VonKries() (*in module colour.corresponding*), 281
CorrespondingChromaticitiesPrediction (*class in colour*), 277
CorrespondingColourDataset (*class in colour*), 276
CQS_Specification (*class in colour.quality*), 641
CRI_Specification (*class in colour.quality*), 640
cube() (*in module colour.plotting*), 636
CV_range() (*in module colour*), 511
CVD_MATRICES_MACHADO2010 (*in module colour*), 107
cvd_matrix_Machado2009() (*in module colour*), 106
cylindrical_to_cartesian() (*in module colour.algebra*), 68

D

data() (*colour.utilities.CaseInsensitiveMapping property*), 698
daylight_locus_function() (*in module colour.colorimetry*), 167
DCDM_XYZ_COLOURSPACE (*in module colour.models*), 407
DCI_P3_COLOURSPACE (*in module colour.models*), 407
DCI_P3_P_COLOURSPACE (*in module colour.models*), 408
default (*colour.NullInterpolator attribute*), 57
default() (*colour.NullInterpolator property*), 57
DEFAULT_FLOAT_DTYPE (*in module colour.constants*), 243
DEFAULT_INT_DTYPE (*in module colour.constants*), 244
DEFAULT_SPECTRAL_SHAPE (*in module colour*), 149
delta_E() (*in module colour*), 283
delta_E_CAM02LCD() (*in module colour.difference*), 289
delta_E_CAM02SCD() (*in module colour.difference*), 290
delta_E_CAM02UCS() (*in module colour.difference*), 291
delta_E_CAM16LCD() (*in module colour.difference*), 292
delta_E_CAM16SCD() (*in module colour.difference*), 292
delta_E_CAM16UCS() (*in module colour.difference*), 293
delta_E_CIE1976() (*in module colour.difference*), 284
delta_E_CIE1994() (*in module colour.difference*), 285
delta_E_CIE2000() (*in module colour.difference*), 286
delta_E_CMC() (*in module colour.difference*), 288
delta_E_DIN99() (*in module colour.difference*), 294
DELTA_E_METHODS (*in module colour*), 284
describe_conversion_path() (*in module colour*), 298
describe_environment() (*in module colour.utilities*), 702
DIN99_to_Lab() (*in module colour*), 365
disable_multiprocessing (*class in colour.utilities*), 668
DISPLAY_P3_COLOURSPACE (*in module colour.models*), 408
DISPLAYS_RGB_PRIMARIES (*in module colour*), 123
domain (*colour.continuous.AbstractContinuousFunction attribute*), 253
domain (*colour.continuous.MultiSignals attribute*), 264
domain (*colour.continuous.Signal attribute*), 256
domain() (*colour.continuous.AbstractContinuousFunction property*), 254
domain() (*colour.continuous.MultiSignals property*), 269
domain() (*colour.continuous.Signal property*), 259
domain_distance() (*colour.continuous.AbstractContinuousFunction method*), 254
domain_range_scale (*class in colour*), 665
dominant_wavelength() (*in module colour*), 204
DON_RGB_4_COLOURSPACE (*in module colour.models*),

408
dot_matrix() (*in module colour.utilities*), 691
dot_vector() (*in module colour.utilities*), 691
DRAGON_COLOR_2_COLOURSPACE (*in module colour.models*), 412
DRAGON_COLOR_COLOURSPACE (*in module colour.models*), 412
dtype (*colour.continuous.MultiSignals attribute*), 264
dtype (*colour.continuous.Signal attribute*), 256
dtype() (*colour.continuous.MultiSignals property*), 269
dtype() (*colour.continuous.Signal property*), 260

E

ECI_RGB_V2_COLOURSPACE (*in module colour.models*), 408
EKTA_SPACE_PS_5_COLOURSPACE (*in module colour.models*), 409
ellipse_coefficients_canonical_form() (*in module colour.algebra*), 72
ellipse_coefficients_general_form() (*in module colour.algebra*), 71
ellipse_fitting() (*in module colour.algebra*), 73
ellipse_fitting_Halir1998() (*in module colour.algebra*), 75
ELLIPSE_FITTING_METHODS (*in module colour.algebra*), 73
end (*colour.SpectralShape attribute*), 124
end() (*colour.SpectralShape property*), 125
eotf() (*in module colour*), 438
eotf_BT1886() (*in module colour.models*), 444
eotf_BT2020() (*in module colour.models*), 445
eotf_DCDM() (*in module colour.models*), 441
eotf_DICOMGSDF() (*in module colour.models*), 442
eotf_HLG_BT2100() (*in module colour.models*), 446
eotf_inverse() (*in module colour*), 439
eotf_inverse_BT1886() (*in module colour.models*), 445
eotf_inverse_DCDM() (*in module colour.models*), 441
eotf_inverse_DICOMGSDF() (*in module colour.models*), 443
eotf_inverse_HLG_BT2100() (*in module colour.models*), 448
eotf_inverse_PQ_BT2100() (*in module colour.models*), 449
eotf_inverse_sRGB() (*in module colour.models*), 453
eotf_inverse_ST2084() (*in module colour.models*), 452
EOTF_INVERSES (*in module colour*), 440
eotf_PQ_BT2100() (*in module colour.models*), 449
eotf_SMPTE240M() (*in module colour.models*), 450
eotf_sRGB() (*in module colour.models*), 453
eotf_ST2084() (*in module colour.models*), 451
EOTFS (*in module colour*), 439
EPSILON (*in module colour.constants*), 244
ERIMM_RGB_COLOURSPACE (*in module colour.models*), 413
euclidean_distance() (*in module colour.algebra*), 69
excitation_purity() (*in module colour*), 206
extend_line_segment() (*in module colour.algebra*), 70
extrapolate() (*colour.MultiSpectralDistributions method*), 138, 141
extrapolate() (*colour.SpectralDistribution method*), 126, 129
Extrapolator (*class in colour*), 51
extrapolator (*colour.continuous.AbstractContinuousFunction attribute*), 253
extrapolator (*colour.continuous.MultiSignals attribute*), 264
extrapolator (*colour.continuous.Signal attribute*), 256
extrapolator() (*colour.continuous.AbstractContinuousFunction property*), 254
extrapolator() (*colour.continuous.MultiSignals property*), 269
extrapolator() (*colour.continuous.Signal property*), 260
extrapolator_args (*colour.continuous.AbstractContinuousFunction attribute*), 253
extrapolator_args (*colour.continuous.MultiSignals attribute*), 264
extrapolator_args (*colour.continuous.Signal attribute*), 257
extrapolator_args() (*colour.continuous.AbstractContinuousFunction property*), 254
extrapolator_args() (*colour.continuous.MultiSignals property*), 269
extrapolator_args() (*colour.continuous.Signal property*), 260

F

F_GAMUT_COLOURSPACE (*in module colour.models*), 409
FAIRCHILD_CAT (*in module colour.adaptation*), 49
fill_nan() (*colour.continuous.AbstractContinuousFunction method*), 253, 255
fill_nan() (*colour.continuous.MultiSignals method*), 264, 269
fill_nan() (*colour.continuous.Signal method*), 257, 260
fill_nan() (*in module colour.utilities*), 694
filter_kwargs() (*in module colour.utilities*), 672
filter_mapping() (*in module colour.utilities*), 672
filter_warnings() (*in module colour.utilities*), 701
first_item() (*in module colour.utilities*), 673
first_key_from_value() (*colour.utilities.Lookup method*), 698
FLOATING_POINT_NUMBER_PATTERN (*in module colour.constants*), 244

from_range_1() (*in module colour.utilities*), 677
from_range_10() (*in module colour.utilities*), 678
from_range_100() (*in module colour.utilities*), 679
from_range_degrees() (*in module colour.utilities*), 680
from_range_int() (*in module colour.utilities*), 680
full_to_legal() (*in module colour*), 509
function (*colour.continuous.AbstractContinuousFunction attribute*), 253
function (*colour.continuous.MultiSignals attribute*), 264
function (*colour.continuous.Signal attribute*), 257
function() (*colour.continuous.AbstractContinuousFunction property*), 255
function() (*colour.continuous.MultiSignals property*), 270
function() (*colour.continuous.Signal property*), 261

G

gamma_function() (*in module colour*), 436
generate_pulse_waves() (*in module colour.volume*), 712
get_domain_range_scale() (*in module colour*), 665
grid() (*in module colour.plotting*), 635

H

handle_numpy_errors() (*in module colour.utilities*), 667
HDR_CIELAB_METHODS (*in module colour*), 381
hdr_CIELab_to_XYZ() (*in module colour*), 380
HDR_IPT_METHODS (*in module colour*), 383
hdr_IPT_to_XYZ() (*in module colour*), 382
header (*colour.SpectralDistribution_IESTM2714 attribute*), 335
header() (*colour.SpectralDistribution_IESTM2714 property*), 336
HEX_to_RGB() (*in module colour.notation*), 531
HSL_to_RGB() (*in module colour*), 518
HSV_to_RGB() (*in module colour*), 516
Hunt_Specification (*class in colour*), 93
HUNT_VIEWING_CONDITIONS (*in module colour*), 94
Hunter_Lab_to_XYZ() (*in module colour*), 361
Hunter_Rdab_to_XYZ() (*in module colour*), 363
HUNTERLAB_ILLUMINANTS (*in module colour*), 203

I

ICTCP_to_RGB() (*in module colour*), 513
ignore_numpy_errors() (*in module colour.utilities*), 667
ignore_python_warnings() (*in module colour.utilities*), 668
ILLUMINANTS (*in module colour*), 202
ILLUMINANTS_OPTIMAL_COLOUR_STIMULI (*in module colour*), 705
ILLUMINANTS_SDS (*in module colour*), 203
ImageAttribute_Specification (*class in colour.io*), 301
in_array() (*in module colour.utilities*), 688
insert() (*colour.LUTSequence method*), 318, 319
INTEGER_THRESHOLD (*in module colour.constants*), 244
intermediate_lightness_function_CIE1976() (*in module colour.colorimetry*), 221
intermediate_luminance_function_CIE1976() (*in module colour.colorimetry*), 227
interpolate() (*colour.MultiSpectralDistributions method*), 138, 142
interpolate() (*colour.SpectralDistribution method*), 126, 130
interpolator (*colour.continuous.AbstractContinuousFunction attribute*), 253
interpolator (*colour.continuous.MultiSignals attribute*), 264
interpolator (*colour.continuous.Signal attribute*), 256
interpolator() (*colour.continuous.AbstractContinuousFunction property*), 255
interpolator() (*colour.continuous.MultiSignals property*), 270
interpolator() (*colour.continuous.Signal property*), 261
interpolator() (*colour.Extrapolator property*), 52
interpolator_args (*colour.continuous.AbstractContinuousFunction attribute*), 253
interpolator_args (*colour.continuous.MultiSignals attribute*), 264
interpolator_args (*colour.continuous.Signal attribute*), 256
interpolator_args() (*colour.continuous.AbstractContinuousFunction property*), 255
interpolator_args() (*colour.continuous.MultiSignals property*), 270
interpolator_args() (*colour.continuous.Signal property*), 261
intersect_line_segments() (*in module colour.algebra*), 70
interval (*colour.SpectralShape attribute*), 124
interval() (*colour.SpectralShape property*), 125
interval() (*in module colour.utilities*), 687
IPT_hue_angle() (*in module colour*), 378
IPT_to_XYZ() (*in module colour*), 377
is_domain_explicit() (*colour.LUT1D method*), 305, 307
is_domain_explicit() (*colour.LUT3D method*), 312, 315
is_domain_explicit() (*colour.LUT3x1D method*), 308, 311
is_identity() (*in module colour.algebra*), 75
is_integer() (*in module colour.utilities*), 671
is_iterable() (*in module colour.utilities*), 670
is_matplotlib_installed() (*in module*

colour.utilities), 669
is_networkx_installed() (*in module colour.utilities*), 669
is_numeric() (*in module colour.utilities*), 671
is_openimageio_installed() (*in module colour.utilities*), 669
is_pandas_installed() (*in module colour.utilities*), 670
is_sibling() (*in module colour.utilities*), 671
is_spow_enabled() (*in module colour.algebra*), 78
is_string() (*in module colour.utilities*), 670
is_uniform() (*colour.continuous.AbstractContinuousFunction method*), 254, 255
is_uniform() (*in module colour.utilities*), 687
is_within_macadam_limits() (*in module colour*), 705
is_within_mesh_volume() (*in module colour*), 706
is_within_pointer_gamut() (*in module colour*), 707
is_within_visible_spectrum() (*in module colour*), 711

J

Jab_to_JCh() (*in module colour.models*), 344
JCh_to_Jab() (*in module colour.models*), 345
JMh_CAM16_to_CAM16LCD() (*in module colour*), 372
JMh_CAM16_to_CAM16SCD() (*in module colour*), 373
JMh_CAM16_to_CAM16UCS() (*in module colour*), 375
JMh_CIECAM02_to_CAM02LCD() (*in module colour*), 367
JMh_CIECAM02_to_CAM02SCD() (*in module colour*), 368
JMh_CIECAM02_to_CAM02UCS() (*in module colour*), 370
JzAzBz_to_XYZ() (*in module colour*), 386

K

K_M (*in module colour.constants*), 242
kernel (*colour.KernelInterpolator attribute*), 53
kernel() (*colour.KernelInterpolator property*), 54
kernel_args (*colour.KernelInterpolator attribute*), 53
kernel_args() (*colour.KernelInterpolator property*), 54
kernel_cardinal_spline() (*in module colour*), 63
kernel_lanczos() (*in module colour*), 63
kernel_linear() (*in module colour*), 62
kernel_nearest_neighbour() (*in module colour*), 61
kernel_sinc() (*in module colour*), 62
KernelInterpolator (*class in colour*), 53
keys_from_value() (*colour.utilities.Lookup method*), 698
KP_M (*in module colour.constants*), 242

L

Lab_to_DIN99() (*in module colour*), 365
Lab_to_LCHab() (*in module colour*), 347
Lab_to_XYZ() (*in module colour*), 346
label_rectangles() (*in module colour.plotting*), 546
labels (*colour.continuous.MultiSignals attribute*), 264
labels() (*colour.continuous.MultiSignals property*), 270
lagrange_coefficients() (*in module colour*), 60
lagrange_coefficients_ASTME2022() (*in module colour.colorimetry*), 186
LCHab_to_Lab() (*in module colour*), 348
LCHuv_to_Luv() (*in module colour*), 351
least_square_mapping_MoorePenrose() (*in module colour.algebra*), 77
LEFS (*in module colour*), 216
left() (*colour.Extrapolator property*), 52
legal_to_full() (*in module colour*), 510
lerp() (*in module colour.utilities*), 694
LIGHT_SOURCES (*in module colour*), 204
LIGHT_SOURCES_SDS (*in module colour*), 204
LIGHT_SPEED (*in module colour.constants*), 243
lightness() (*in module colour*), 217
lightness_CIE1976() (*in module colour.colorimetry*), 220
lightness_Fairchild2010() (*in module colour.colorimetry*), 222
lightness_Fairchild2011() (*in module colour.colorimetry*), 223
lightness_Glasser1958() (*in module colour.colorimetry*), 219
LIGHTNESS_METHODS (*in module colour*), 218
lightness_Wyszecki1963() (*in module colour.colorimetry*), 220
linear_conversion() (*in module colour.utilities*), 693
linear_function() (*in module colour*), 437
linear_table() (*colour.LUT1D method*), 305
linear_table() (*colour.LUT1D static method*), 308
linear_table() (*colour.LUT3D method*), 313
linear_table() (*colour.LUT3D static method*), 315
linear_table() (*colour.LUT3x1D method*), 308
linear_table() (*colour.LUT3x1D static method*), 311
LinearInterpolator (*class in colour*), 55
LineSegmentsIntersections_Specification (*class in colour.algebra*), 74
LLAB_InductionFactors (*class in colour.appearance*), 97
LLAB_Specification (*class in colour*), 95
LLAB_VIEWING_CONDITIONS (*in module colour*), 96
LMS_10_degree_cmfs_to_XYZ_10_degree_cmfs() (*in module colour.colorimetry*), 201
LMS_2_degree_cmfs_to_XYZ_2_degree_cmfs() (*in module colour.colorimetry*), 201
LMS_CMFS (*in module colour*), 197
LMS_ConeFundamentals (*class in colour.colorimetry*), 195
LOG3G10_DECODING_METHODS (*in module*

colour.models), 483
LOG3G10_ENCODING_METHODS (*in module colour.models*), 481
log_decoding() (*in module colour*), 462
log_decoding_ACEScc() (*in module colour.models*), 466
log_decoding_ACEScct() (*in module colour.models*), 467
log_decoding_ACESproxy() (*in module colour.models*), 469
log_decoding_ALEXALogC() (*in module colour.models*), 471
log_decoding_CanonLog() (*in module colour.models*), 476
log_decoding_CanonLog2() (*in module colour.models*), 472
log_decoding_CanonLog3() (*in module colour.models*), 474
log_decoding_Cineon() (*in module colour.models*), 477
log_decoding_ERIMMRGB() (*in module colour.models*), 479
log_decoding_FLog() (*in module colour.models*), 481
log_decoding_Log3G10() (*in module colour.models*), 483
log_decoding_Log3G12() (*in module colour.models*), 484
log_decoding_Panalog() (*in module colour.models*), 486
log_decoding_PivotedLog() (*in module colour.models*), 487
log_decoding_Protune() (*in module colour.models*), 489
log_decoding_REDLog() (*in module colour.models*), 490
log_decoding_REDLogFilm() (*in module colour.models*), 491
log_decoding_SLog() (*in module colour.models*), 493
log_decoding_SLog2() (*in module colour.models*), 494
log_decoding_SLog3() (*in module colour.models*), 496
log_decoding_ViperLog() (*in module colour.models*), 499
log_decoding_VLog() (*in module colour.models*), 498
LOG_DECODINGS (*in module colour*), 463
log_encoding() (*in module colour*), 460
log_encoding_ACEScc() (*in module colour.models*), 465
log_encoding_ACEScct() (*in module colour.models*), 467
log_encoding_ACESproxy() (*in module colour.models*), 468
log_encoding_ALEXALogC() (*in module colour.models*), 470
log_encoding_CanonLog() (*in module colour.models*), 475
log_encoding_CanonLog2() (*in module colour.models*), 471
log_encoding_CanonLog3() (*in module colour.models*), 473
log_encoding_Cineon() (*in module colour.models*), 477
log_encoding_ERIMMRGB() (*in module colour.models*), 478
log_encoding_FLog() (*in module colour.models*), 480
log_encoding_Log3G10() (*in module colour.models*), 482
log_encoding_Log3G12() (*in module colour.models*), 484
log_encoding_Panalog() (*in module colour.models*), 485
log_encoding_PivotedLog() (*in module colour.models*), 487
log_encoding_Protune() (*in module colour.models*), 488
log_encoding_REDLog() (*in module colour.models*), 489
log_encoding_REDLogFilm() (*in module colour.models*), 490
log_encoding_SLog() (*in module colour.models*), 492
log_encoding_SLog2() (*in module colour.models*), 493
log_encoding_SLog3() (*in module colour.models*), 495
log_encoding_ViperLog() (*in module colour.models*), 499
log_encoding_VLog() (*in module colour.models*), 497
LOG_ENCODINGS (*in module colour*), 462
Lookup (*class in colour.utilities*), 698
lower_items() (*colour.utilities.CaseInsensitiveMapping method*), 697, 698
lower_items() (*in module colour*), 506
luminance() (*in module colour*), 224
luminance_ASTMD1535() (*in module colour.colorimetry*), 228
luminance_CIE1976() (*in module colour.colorimetry*), 226
luminance_Fairchild2010() (*in module colour.colorimetry*), 229
luminance_Fairchild2011() (*in module colour.colorimetry*), 230
LUMINANCE_METHODS (*in module colour*), 225
luminance_Newhall1943() (*in module colour.colorimetry*), 226
luminous_efficacy() (*in module colour*), 207
luminous_efficiency() (*in module colour*), 208
luminous_flux() (*in module colour*), 208
LUT1D (*class in colour*), 305
LUT3D (*class in colour*), 312

LUT3x1D (*class in colour*), 308
LUT_to_LUT() (*in module colour.io*), 323
LUTSequence (*class in colour*), 318
Luv_to_LCHuv() (*in module colour*), 351
Luv_to_uv() (*in module colour*), 352
Luv_to_XYZ() (*in module colour*), 350
Luv_uv_to_xy() (*in module colour*), 354

M

mapping (*colour.SpectralDistribution_IESTM2714 attribute*), 335
mapping() (*colour.SpectralDistribution_IESTM2714 property*), 336
MAX_RGB_COLOURSPACE (*in module colour.models*), 409
maximum_angular_size_Barten1999() (*in module colour.contrast*), 252
message_box() (*in module colour.utilities*), 699
method() (*colour.Extrapolator property*), 52
metric_mse() (*in module colour.utilities*), 695
metric_psnr() (*in module colour.utilities*), 696
MULTI_SD_TO_XYZ_METHODS (*in module colour*), 178
multi_sds_to_XYZ() (*in module colour*), 176
multi_sds_to_XYZ_ASTME308() (*in module colour.colorimetry*), 182
multi_sds_to_XYZ_integration() (*in module colour.colorimetry*), 190
multi_signals_unpack_data() (*colour.continuous.MultiSignals method*), 264
multi_signals_unpack_data() (*colour.continuous.MultiSignals static method*), 271
multiprocessing_pool() (*in module colour.utilities*), 669
MultiSignals (*class in colour.continuous*), 263
MultiSpectralDistributions (*class in colour*), 138
munsell_colour_to_xyY() (*in module colour*), 522
MUNSELL_COLOURS (*in module colour*), 523
munsell_value() (*in module colour*), 524
munsell_value_ASTMD1535() (*in module colour.notation*), 530
munsell_value_Ladd1955() (*in module colour.notation*), 528
munsell_value_McCamy1987() (*in module colour.notation*), 529
MUNSELL_VALUE_METHODS (*in module colour*), 525
munsell_value_Moon1943() (*in module colour.notation*), 527
munsell_value_Munsell1933() (*in module colour.notation*), 526
munsell_value_Priest1920() (*in module colour.notation*), 525
munsell_value_Saunderson1944() (*in module colour.notation*), 527

N

name (*colour.continuous.AbstractContinuousFunction attribute*), 253
name (*colour.RGB_Colourspace attribute*), 397
name (*in module colour.models*), 415
name() (*colour.continuous.AbstractContinuousFunction property*), 255
name() (*colour.RGB_Colourspace property*), 399
Nayatani95_Specification (*class in colour*), 99
ndarray_write() (*in module colour.utilities*), 695
NearestNeighbourInterpolator (*class in colour*), 55
normalise() (*colour.MultiSpectralDistributions method*), 139, 145
normalise() (*colour.SpectralDistribution method*), 127, 135
normalise_maximum() (*in module colour.utilities*), 686
normalise_vector() (*in module colour.algebra*), 69
normalised_primary_matrix() (*in module colour*), 393
NTSC_1953_COLOURSPACE (*in module colour.models*), 410
NTSC_1987_COLOURSPACE (*in module colour.models*), 410
NullInterpolator (*class in colour*), 56
numpy_print_options() (*in module colour.utilities*), 702

O

oetf() (*in module colour*), 426
oetf_ARIBSTDB67() (*in module colour.models*), 428
oetf_BT2020() (*in module colour.models*), 430
oetf_BT601() (*in module colour.models*), 433
oetf_BT709() (*in module colour.models*), 434
oetf_HLG_BT2100() (*in module colour.models*), 430
oetf_inverse() (*in module colour*), 427
oetf_inverse_ARIBSTDB67() (*in module colour.models*), 429
oetf_inverse_BT601() (*in module colour.models*), 434
oetf_inverse_BT709() (*in module colour.models*), 435
oetf_inverse_HLG_BT2100() (*in module colour.models*), 431
oetf_inverse_PQ_BT2100() (*in module colour.models*), 432
OETF_INVERSES (*in module colour*), 427
oetf_PQ_BT2100() (*in module colour.models*), 432
oetf_SMPTE240M() (*in module colour.models*), 435
OETFS (*in module colour*), 427
ootf() (*in module colour*), 454
ootf_HLG_BT2100() (*in module colour.models*), 456
ootf_inverse() (*in module colour*), 455
ootf_inverse_HLG_BT2100() (*in module colour.models*), 458
ootf_inverse_PQ_BT2100() (*in module colour.models*), 459
OOTF_INVERSES (*in module colour*), 456
ootf_PQ_BT2100() (*in module colour.models*), 459

OOTFS (*in module colour*), 455
optical_MTF_Barten1999() (*in module colour.contrast*), 249
orient() (*in module colour.utilities*), 692
OSA_UCS_to_XYZ() (*in module colour*), 384

P

P3_D65_COLOURSPACE (*in module colour.models*), 410
padding_args (*colour.KernelInterpolator attribute*), 53
padding_args() (*colour.KernelInterpolator property*), 54
PAL_SECAM_COLOURSPACE (*in module colour.models*), 410
path (*colour.SpectralDistribution_IESTM2714 attribute*), 335
path() (*colour.SpectralDistribution_IESTM2714 property*), 336
PchipInterpolator (*class in colour*), 58
PHOTOPIC_LEFS (*in module colour*), 217
PLANCK_CONSTANT (*in module colour.constants*), 243
planck_law() (*in module colour.colorimetry*), 173
plot_automatic_colour_conversion_graph() (*in module colour.plotting*), 637
plot_blackbody_colours() (*in module colour.plotting*), 564
plot_blackbody_spectral_radiance() (*in module colour.plotting*), 563
plot_chromaticity_diagram() (*in module colour.plotting.diagrams*), 583
plot_chromaticity_diagram_CIE1931() (*in module colour.plotting*), 570
plot_chromaticity_diagram_CIE1960UCS() (*in module colour.plotting*), 571
plot_chromaticity_diagram_CIE1976UCS() (*in module colour.plotting*), 572
plot_chromaticity_diagram_colours() (*in module colour.plotting.diagrams*), 581
plot_colour_quality_bars() (*in module colour.plotting.quality*), 620
plot_constant_hue_loci() (*in module colour.plotting*), 602
plot_corresponding_chromaticities_prediction() (*in module colour.plotting*), 568
plot_cvd_simulation_Machado2009() (*in module colour.plotting*), 565
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1931() (*in module colour.plotting*), 596
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1960UCS() (*in module colour.plotting*), 597
plot_ellipses_MacAdam1942_in_chromaticity_diagram_CIE1976UCS() (*in module colour.plotting*), 599
plot_image() (*in module colour.plotting*), 551
plot_multi_cctfs() (*in module colour.plotting*), 601
plot_multi_cmfs() (*in module colour.plotting*), 555
plot_multi_colour_checkers() (*in module colour.plotting*), 567
plot_multi_colour_swatches() (*in module colour.plotting*), 547
plot_multi_functions() (*in module colour.plotting*), 550
plot_multi_illuminant_sds() (*in module colour.plotting*), 557
plot_multi_lightness_functions() (*in module colour.plotting*), 560
plot_multi_luminance_functions() (*in module colour.plotting*), 562
plot_multi_munsell_value_functions() (*in module colour.plotting*), 611
plot_multi_sds() (*in module colour.plotting*), 553
plot_multi_sds_colour_quality_scales_bars() (*in module colour.plotting*), 619
plot_multi_sds_colour_rendering_indexes_bars() (*in module colour.plotting*), 616
plot_planckian_locus() (*in module colour.plotting.temperature*), 626
plot_planckian_locus_CIE1931() (*in module colour.plotting.temperature*), 627
plot_planckian_locus_CIE1960UCS() (*in module colour.plotting.temperature*), 628
plot_planckian_locus_in_chromaticity_diagram() (*in module colour.plotting.temperature*), 629
plot_planckian_locus_in_chromaticity_diagram_CIE1931() (*in module colour.plotting*), 622
plot_planckian_locus_in_chromaticity_diagram_CIE1960UCS() (*in module colour.plotting*), 623
plot_pointer_gamut() (*in module colour.plotting.models*), 606
plot_RGB_chromaticities_in_chromaticity_diagram() (*in module colour.plotting.models*), 608
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1931() (*in module colour.plotting*), 591
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1960UCS() (*in module colour.plotting*), 593
plot_RGB_chromaticities_in_chromaticity_diagram_CIE1976UCS() (*in module colour.plotting*), 594
plot_RGB_colourspaces_gamuts() (*in module colour.plotting*), 632
plot_RGB_colourspaces_in_chromaticity_diagram() (*in module colour.plotting.models*), 607
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1931() (*in module colour.plotting*), 587
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1960UCS() (*in module colour.plotting*), 588
plot_RGB_colourspaces_in_chromaticity_diagram_CIE1976UCS() (*in module colour.plotting*), 590
plot_RGB_scatter() (*in module colour.plotting*), 633
plot_sds_in_chromaticity_diagram() (*in module colour.plotting.diagrams*), 584
plot_sds_in_chromaticity_diagram_CIE1931() (*in module colour.plotting*), 574

plot_sds_in_chromaticity_diagram_CIE1960UCS() (*in module colour.plotting*), 575
plot_sds_in_chromaticity_diagram_CIE1976UCS() (*in module colour.plotting*), 577
plot_single_cctf() (*in module colour.plotting*), 600
plot_single_cmfs() (*in module colour.plotting*), 555
plot_single_colour_checker() (*in module colour.plotting*), 566
plot_single_colour_swatch() (*in module colour.plotting*), 546
plot_single_function() (*in module colour.plotting*), 549
plot_single_illuminant_sd() (*in module colour.plotting*), 556
plot_single_lightness_function() (*in module colour.plotting*), 559
plot_single_luminance_function() (*in module colour.plotting*), 561
plot_single_munsell_value_function() (*in module colour.plotting*), 610
plot_single_sd() (*in module colour.plotting*), 552
plot_single_sd_colour_quality_scale_bars() (*in module colour.plotting*), 617
plot_single_sd_colour_rendering_index_bars() (*in module colour.plotting*), 614
plot_single_sd_rayleigh_scattering() (*in module colour.plotting*), 612
plot_spectral_locus() (*in module colour.plotting.diagrams*), 580
plot_the_blue_sky() (*in module colour.plotting*), 613
plot_visible_spectrum() (*in module colour.plotting*), 558
point_at_angle_on_ellipse() (*in module colour.algebra*), 72
POINTER_GAMUT_BOUNDARIES (*in module colour*), 521
POINTER_GAMUT_DATA (*in module colour*), 521
POINTER_GAMUT_ILLUMINANT (*in module colour*), 521
polar_to_cartesian() (*in module colour.algebra*), 67
polynomial_expansion() (*in module colour*), 108
polynomial_expansion_Finlayson2015() (*in module colour.characterisation*), 114
POLYNOMIAL_EXPANSION_METHODS (*in module colour*), 108
polynomial_expansion_Vandermonde() (*in module colour.characterisation*), 114
primaries (*colour.RGB_Colourspace attribute*), 397
primaries (*in module colour.models*), 415
primaries() (*colour.RGB_Colourspace property*), 400
primaries_whitepoint() (*in module colour*), 394
print_numpy_errors() (*in module colour.utilities*), 667
Prismatic_to_RGB() (*in module colour*), 515
PROPHOTO_RGB_COLOURSPACE (*in module colour.models*), 413
PROTUNE_NATIVE_COLOURSPACE (*in module colour.models*), 409
pupil_diameter_Barten1999() (*in module colour.contrast*), 250

Q

quad() (*in module colour.plotting*), 635

R

raise_numpy_errors() (*in module colour.utilities*), 667
random_triplet_generator() (*in module colour.algebra*), 76
range (*colour.continuous.AbstractContinuousFunction attribute*), 253
range (*colour.continuous.MultiSignals attribute*), 264
range (*colour.continuous.Signal attribute*), 256
range() (*colour.continuous.AbstractContinuousFunction property*), 256
range() (*colour.continuous.MultiSignals property*), 273
range() (*colour.continuous.Signal property*), 261
range() (*colour.SpectralShape method*), 125
rayleigh_optical_depth() (*in module colour.phenomena*), 542
rayleigh_scattering() (*in module colour*), 532
reaction_rate_MichealisMenten() (*in module colour.biochemistry*), 103
read() (*colour.SpectralDistribution_IESTM2714 method*), 335, 336
read_image() (*in module colour*), 299
read_image_Imageio() (*in module colour.io*), 304
READ_IMAGE_METHODS (*in module colour*), 299
read_image_OpenImageIO() (*in module colour.io*), 302
read_LUT() (*in module colour*), 320
read_LUT_Cinespace() (*in module colour.io*), 324
read_LUT_IridasCube() (*in module colour.io*), 325
read_LUT_SonySPI1D() (*in module colour.io*), 327
read_LUT_SonySPI3D() (*in module colour.io*), 329
read_sds_from_csv_file() (*in module colour*), 330
read_sds_from_xrite_file() (*in module colour*), 337
read_spectral_data_from_csv_file() (*in module colour*), 332
RED_COLOR_2_COLOURSPACE (*in module colour.models*), 411
RED_COLOR_3_COLOURSPACE (*in module colour.models*), 411
RED_COLOR_4_COLOURSPACE (*in module colour.models*), 411
RED_COLOR_COLOURSPACE (*in module colour.models*), 410

RED_WIDE_GAMUT_RGB_COLOURSPACE (*in module colour.models*), 411
reflection_geometry (*colour.SpectralDistribution_IESTM2714 attribute*), 335
reflection_geometry() (*colour.SpectralDistribution_IESTM2714 property*), 336
relative_tolerance (*colour.NullInterpolator attribute*), 57
relative_tolerance() (*colour.NullInterpolator property*), 57
render() (*in module colour.plotting*), 545
retinal_illuminance_Barten1999() (*in module colour.contrast*), 251
RGB_10_degree_cmfs_to_LMS_10_degree_cmfs() (*in module colour.colorimetry*), 200
RGB_10_degree_cmfs_to_XYZ_10_degree_cmfs() (*in module colour.colorimetry*), 199
RGB_2_degree_cmfs_to_XYZ_2_degree_cmfs() (*in module colour.colorimetry*), 199
RGB_CMFS (*in module colour*), 198
RGB_ColourMatchingFunctions (*class in colour.colorimetry*), 196
RGB_Colourspace (*class in colour*), 395
RGB_colourspace_limits() (*in module colour*), 708
RGB_colourspace_pointer_gamut_coverage_MonteCarlo() (*in module colour*), 708
RGB_colourspace_visible_spectrum_coverage_MonteCarlo() (*in module colour*), 709
RGB_colourspace_volume_coverage_MonteCarlo() (*in module colour*), 710
RGB_colourspace_volume_MonteCarlo() (*in module colour*), 709
RGB_COLOURSPACES (*in module colour*), 401
RGB_DisplayPrimaries (*class in colour.characterisation*), 122
RGB_luminance() (*in module colour*), 394
RGB_luminance_equation() (*in module colour*), 395
RGB_SpectralSensitivities (*class in colour.characterisation*), 121
RGB_to_CMY() (*in module colour*), 519
RGB_to_HEX() (*in module colour.notation*), 531
RGB_to_HSL() (*in module colour*), 517
RGB_to_HSV() (*in module colour*), 516
RGB_to_ICTCP() (*in module colour*), 512
RGB_to_Prismatic() (*in module colour*), 514
RGB_to_RGB() (*in module colour*), 389
RGB_to_RGB_matrix() (*in module colour*), 390
RGB_to_sd_Smits1999() (*in module colour.recovery*), 647
RGB_to_XYZ() (*in module colour*), 388
RGB_to_XYZ_matrix (*colour.RGB_Colourspace attribute*), 397
RGB_to_XYZ_matrix (*in module colour.models*), 415
RGB_to_XYZ_matrix() (*colour.RGB_Colourspace property*), 398
RGB_to_YCbCr() (*in module colour*), 502
RGB_to_YcCbcCrc() (*in module colour*), 507
RGB_to_YCoCg() (*in module colour*), 511
right() (*colour.Extrapolator property*), 52
RIMM_RGB_COLOURSPACE (*in module colour.models*), 412
RLAB_D_FACTOR (*in module colour*), 101
RLAB_Specification (*class in colour*), 101
RLAB_VIEWING_CONDITIONS (*in module colour*), 102
ROMM_RGB_COLOURSPACE (*in module colour.models*), 412
row_as_diagonal() (*in module colour.utilities*), 690
RUSSELL_RGB_COLOURSPACE (*in module colour.models*), 413

S

S_GAMUT3_CINE_COLOURSPACE (*in module colour.models*), 417
S_GAMUT3_COLOURSPACE (*in module colour.models*), 417
S_GAMUT_COLOURSPACE (*in module colour.models*), 416
scattering_cross_section() (*in module colour*), 541
SCOTOPIC_LEFS (*in module colour*), 217
sd_blackbody() (*in module colour*), 154
sd_CIE_illuminant_D_series() (*in module colour*), 152
sd_CIE_standard_illuminant_A() (*in module colour*), 151
sd_constant() (*in module colour*), 162
sd_gaussian() (*in module colour*), 163
sd_gaussian_fwhm() (*in module colour.colorimetry*), 168
SD_GAUSSIAN_METHODS (*in module colour*), 163
sd_gaussian_normal() (*in module colour.colorimetry*), 168
sd_mesopic_luminous_efficiency_function() (*in module colour*), 209
sd_multi_leds() (*in module colour*), 165
SD_MULTI_LEDS_METHODS (*in module colour*), 165
sd_multi_leds_Ohno2005() (*in module colour.colorimetry*), 170
sd_ones() (*in module colour*), 162
sd_rayleigh_scattering() (*in module colour*), 533
sd_single_led() (*in module colour*), 164
SD_SINGLE_LED_METHODS (*in module colour*), 164
sd_single_led_Ohno2005() (*in module colour.colorimetry*), 169
sd_to_aces_relative_exposure_values() (*in module colour*), 500
sd_to_XYZ() (*in module colour*), 174
sd_to_XYZ_ASTME308() (*in module colour.colorimetry*), 180
sd_to_XYZ_integration() (*in module colour.colorimetry*), 189

SD_TO_XYZ_METHODS (*in module colour*), 176
sd_to_XYZ_tristimulus_weighting_factors_ASTME308() (*in module colour.colorimetry*), 184
sd_zeros() (*in module colour*), 163
sds_and_multi_sds_to_multi_sds() (*in module colour.colorimetry*), 172
sds_and_multi_sds_to_sds() (*in module colour.colorimetry*), 171
sequence (*colour.LUTSequence attribute*), 318
sequence() (*colour.LUTSequence property*), 320
set_domain_range_scale() (*in module colour*), 666
set_spow_enable() (*in module colour.algebra*), 78
shape (*colour.MultiSpectralDistributions attribute*), 138
shape (*colour.SpectralDistribution attribute*), 126
shape() (*colour.MultiSpectralDistributions property*), 146
shape() (*colour.SpectralDistribution property*), 135
SHARP_CAT (*in module colour.adaptation*), 49
sigma_Barten1999() (*in module colour.contrast*), 250
Signal (*class in colour.continuous*), 256
signal_type (*colour.continuous.MultiSignals attribute*), 264
signal_type() (*colour.continuous.MultiSignals property*), 273
signal_unpack_data() (*colour.continuous.Signal method*), 257
signal_unpack_data() (*colour.continuous.Signal static method*), 262
signals (*colour.continuous.MultiSignals attribute*), 264
signals() (*colour.continuous.MultiSignals property*), 274
SMITS_1999_SDS (*in module colour.recovery*), 647
SMPTE_240M_COLOURSPACE (*in module colour.models*), 413
SMPTE_C_COLOURSPACE (*in module colour.models*), 414
spectral_quantity (*colour.SpectralDistribution_IESTM2714 attribute*), 335
spectral_quantity() (*colour.SpectralDistribution_IESTM2714 property*), 336
spectral_similarity_index() (*in module colour*), 643
SpectralDistribution (*class in colour*), 126
SpectralDistribution_IESTM2714 (*class in colour*), 334
SpectralShape (*class in colour*), 124
spherical_to_cartesian() (*in module colour.algebra*), 66
spow() (*in module colour.algebra*), 79
spow_enable (*class in colour.algebra*), 78
SPRAGUE_C_COEFFICIENTS (*colour.SpragueInterpolator attribute*), 59
SpragueInterpolator (*class in colour*), 58
sRGB_COLOURSPACE (*in module colour.models*), 417
sRGB_to_XYZ() (*in module colour*), 391
STANDARD_OBSERVERS_CMFS (*in module colour*), 198
start (*colour.SpectralShape attribute*), 124
start() (*colour.SpectralShape property*), 126
strict_labels (*colour.MultiSpectralDistributions attribute*), 138
strict_labels() (*colour.MultiSpectralDistributions property*), 147
strict_name (*colour.MultiSpectralDistributions attribute*), 138
strict_name (*colour.SpectralDistribution attribute*), 126
strict_name() (*colour.MultiSpectralDistributions property*), 147
strict_name() (*colour.SpectralDistribution property*), 136
Structure (*class in colour.utilities*), 699
substrate_concentration_MichealisMenten() (*in module colour.biochemistry*), 103
suppress_warnings() (*in module colour.utilities*), 701

T

table_interpolation() (*in module colour*), 60
TABLE_INTERPOLATION_METHODS (*in module colour*), 60
table_interpolation_tetrahedral() (*in module colour.algebra*), 65
table_interpolation_trilinear() (*in module colour.algebra*), 64
to_dataframe() (*colour.continuous.MultiSignals method*), 264, 274
to_domain_1() (*in module colour.utilities*), 673
to_domain_10() (*in module colour.utilities*), 674
to_domain_100() (*in module colour.utilities*), 675
to_domain_degrees() (*in module colour.utilities*), 676
to_domain_int() (*in module colour.utilities*), 677
to_sds() (*colour.MultiSpectralDistributions method*), 139, 147
to_series() (*colour.continuous.Signal method*), 257, 263
transmission_geometry (*colour.SpectralDistribution_IESTM2714 attribute*), 335
transmission_geometry() (*colour.SpectralDistribution_IESTM2714 property*), 337
trim() (*colour.MultiSpectralDistributions method*), 139, 148
trim() (*colour.SpectralDistribution method*), 127, 136
tristimulus_weighting_factors_ASTME2022() (*in module colour.colorimetry*), 187
tsplit() (*in module colour.utilities*), 689
tstack() (*in module colour.utilities*), 688

U

UCS_to_uv() (*in module colour*), 356
UCS_to_XYZ() (*in module colour*), 355
UCS_uv_to_xy() (*in module colour*), 357
uniform_axes3d() (*in module colour.plotting*), 546
use_derived_RGB_to_XYZ_matrix (*colour.RGB_Colourspace attribute*), 397
use_derived_RGB_to_XYZ_matrix (*in module colour.models*), 415
use_derived_RGB_to_XYZ_matrix() (*colour.RGB_Colourspace property*), 400
use_derived_transformation_matrices() (*colour.RGB_Colourspace method*), 397, 400
use_derived_transformation_matrices() (*in module colour.models*), 415
use_derived_XYZ_to_RGB_matrix (*colour.RGB_Colourspace attribute*), 397
use_derived_XYZ_to_RGB_matrix (*in module colour.models*), 415
use_derived_XYZ_to_RGB_matrix() (*colour.RGB_Colourspace property*), 400
uv_to_CCT() (*in module colour*), 651
uv_to_CCT_Krystek1985() (*in module colour.temperature*), 656
UV_TO_CCT_METHODS (*in module colour*), 651
uv_to_CCT_Ohno2013() (*in module colour.temperature*), 658
uv_to_CCT_Robertson1968() (*in module colour.temperature*), 655
uv_to_Luv() (*in module colour*), 353
uv_to_UCS() (*in module colour*), 357
UVW_to_XYZ() (*in module colour*), 359

V

V_GAMUT_COLOURSPACE (*in module colour.models*), 417
values (*colour.MultiSpectralDistributions attribute*), 138
values (*colour.SpectralDistribution attribute*), 126
values() (*colour.MultiSpectralDistributions property*), 149
values() (*colour.SpectralDistribution property*), 137
VON_KRIES_CAT (*in module colour.adaptation*), 49

W

warn_numpy_errors() (*in module colour.utilities*), 667
warning() (*in module colour.utilities*), 700
wavelength_to_XYZ() (*in module colour*), 179
wavelengths (*colour.MultiSpectralDistributions attribute*), 138
wavelengths (*colour.SpectralDistribution attribute*), 126
wavelengths() (*colour.MultiSpectralDistributions property*), 149
wavelengths() (*colour.SpectralDistribution property*), 137
whiteness() (*in module colour*), 231
whiteness_ASTME313() (*in module colour.colorimetry*), 235
whiteness_Berger1959() (*in module colour.colorimetry*), 232
whiteness_CIE2004() (*in module colour.colorimetry*), 237
whiteness_Ganz1979() (*in module colour.colorimetry*), 236
WHITENESS_METHODS (*in module colour*), 232
whiteness_Stensby1968() (*in module colour.colorimetry*), 234
whiteness_Taube1960() (*in module colour.colorimetry*), 233
whitepoint (*colour.RGB_Colourspace attribute*), 397
whitepoint (*in module colour.models*), 415
whitepoint() (*colour.RGB_Colourspace property*), 400
whitepoint_name (*colour.RGB_Colourspace attribute*), 397
whitepoint_name (*in module colour.models*), 415
whitepoint_name() (*colour.RGB_Colourspace property*), 400
window (*colour.KernelInterpolator attribute*), 53
window() (*colour.KernelInterpolator property*), 54
write() (*colour.SpectralDistribution_IESTM2714 method*), 335, 337
write_image() (*in module colour*), 300
write_image_Imageio() (*in module colour.io*), 304
WRITE_IMAGE_METHODS (*in module colour*), 300
write_image_OpenImageIO() (*in module colour.io*), 303
write_LUT() (*in module colour*), 321
write_LUT_Cinespace() (*in module colour.io*), 324
write_LUT_IridasCube() (*in module colour.io*), 326
write_LUT_SonySPI1D() (*in module colour.io*), 328
write_LUT_SonySPI3D() (*in module colour.io*), 329
write_sds_to_csv_file() (*in module colour*), 333

X

x (*colour.KernelInterpolator attribute*), 53
x (*colour.LinearInterpolator attribute*), 55
x (*colour.NullInterpolator attribute*), 56
x (*colour.SpragueInterpolator attribute*), 58
x() (*colour.KernelInterpolator property*), 55
x() (*colour.LinearInterpolator property*), 56
x() (*colour.NullInterpolator property*), 57
x() (*colour.SpragueInterpolator property*), 59
XTREME_RGB_COLOURSPACE (*in module colour.models*), 418
xy_to_CCT() (*in module colour*), 653

xy_to_CCT_CIE_D() (*in module colour.temperature*), 663
xy_to_CCT_Hernandez1999() (*in module colour.temperature*), 661
xy_to_CCT_Kang2002() (*in module colour.temperature*), 662
xy_to_CCT_McCamy1992() (*in module colour.temperature*), 659
XY_TO_CCT_METHODS (*in module colour*), 653
xy_to_Luv_uv() (*in module colour*), 354
xy_to_UCS_uv() (*in module colour*), 358
xy_to_xyY() (*in module colour*), 343
xy_to_XYZ() (*in module colour*), 341
xyY_to_munsell_colour() (*in module colour*), 523
xyY_to_xy() (*in module colour*), 342
xyY_to_XYZ() (*in module colour*), 340
XYZ_ColourMatchingFunctions (*class in colour.colorimetry*), 196
XYZ_outer_surface() (*in module colour.volume*), 713
XYZ_SCALING_CAT (*in module colour.adaptation*), 49
XYZ_to_ATD95() (*in module colour*), 80
XYZ_to_CAM16() (*in module colour*), 87
XYZ_to_CIECAM02() (*in module colour*), 82
XYZ_to_hdr_CIELab() (*in module colour*), 379
XYZ_to_hdr_IPT() (*in module colour*), 381
XYZ_to_Hunt() (*in module colour*), 91
XYZ_to_Hunter_Lab() (*in module colour*), 360
XYZ_to_Hunter_Rdab() (*in module colour*), 363
XYZ_to_IPT() (*in module colour*), 376
XYZ_to_JzAzBz() (*in module colour*), 385
XYZ_to_K_ab_HunterLab1966() (*in module colour*), 362
XYZ_to_Lab() (*in module colour*), 346
XYZ_to_LLAB() (*in module colour*), 94
XYZ_to_Luv() (*in module colour*), 349
XYZ_to_Nayatani95() (*in module colour*), 98
XYZ_to_OSA_UCS() (*in module colour*), 383
XYZ_to_RGB() (*in module colour*), 387
XYZ_to_RGB_matrix (*colour.RGB_Colourspace attribute*), 397
XYZ_to_RGB_matrix (*in module colour.models*), 415
XYZ_to_RGB_matrix() (*colour.RGB_Colourspace property*), 398
XYZ_to_RLAB() (*in module colour*), 100
XYZ_to_sd() (*in module colour*), 644
XYZ_to_sd_Meng2015() (*in module colour.recovery*), 648
XYZ_TO_SD_METHODS (*in module colour*), 646
XYZ_to_sRGB() (*in module colour*), 391
XYZ_to_UCS() (*in module colour*), 355
XYZ_to_UVW() (*in module colour*), 358
XYZ_to_xy() (*in module colour*), 341
XYZ_to_xyY() (*in module colour*), 339

Y

y (*colour.KernelInterpolator attribute*), 53
y (*colour.LinearInterpolator attribute*), 55
y (*colour.NullInterpolator attribute*), 56
y (*colour.PchipInterpolator attribute*), 58
y (*colour.SpragueInterpolator attribute*), 58
y() (*colour.KernelInterpolator property*), 55
y() (*colour.LinearInterpolator property*), 56
y() (*colour.NullInterpolator property*), 57
y() (*colour.PchipInterpolator property*), 58
y() (*colour.SpragueInterpolator property*), 59
YCbCr_to_RGB() (*in module colour*), 504
YCBCR_WEIGHTS (*in module colour*), 506
YcCbcCrc_to_RGB() (*in module colour*), 508
YCoCg_to_RGB() (*in module colour*), 512
yellowness() (*in module colour*), 239
yellowness_ASTMD1925() (*in module colour.colorimetry*), 240
yellowness_ASTME313() (*in module colour.colorimetry*), 241
YELLOWNESS_METHODS (*in module colour*), 239